

Чат-боты

Аннотация

В этом цикле дополнительных занятий мы поговорим о создании чат-ботов для различных популярных платформ и начнем, пожалуй, с самой популярной социальной сети в России и странах СНГ — vk.com.

Про чат-ботов

Многие платформы начали предоставлять разработчикам более продвинутое API, которое позволяет практически встроить свое приложение в эту платформу прозрачно для пользователя. При таком подходе пользователь получает функциональность вашего приложения, не покидая привычного интерфейса сервиса или социальной сети.

Последние годы такой подход набрал очень большую популярность, количество платформ с качественным API для разработки своих ботов растет. Сегодня мы рассмотрим API самой популярной социальной сети в России и странах СНГ — vk.com.

API для vk.com и для многих подобных платформ можно условно разделить на две части:

- Пользовательское API
- API ботов

Когда мы выполняем некоторое обращение к пользовательскому API, мы авторизуемся и выполняем действия от лица этого пользователя, то есть все изменения будут выполнены как бы руками самого пользователя: он будет автором сообщения, записи на стене, загруженного изображения и т. д. Такой подход позволяет создавать программы, имитирующие реальных людей или альтернативные клиенты для платформы. Чтобы снизить количество мошенников, которые разрабатывают «поддельных» пользователей, разработчики обычно накладывают ограничение на использование такого типа API, например, делают некоторые методы недоступными (совсем или без дополнительного подтверждения вашей личности).

API ботов, как правило, не имеют таких ограничений, но пользователь будет обязательно осведомлен, что он общается не с человеком, а с некоторой сущностью внутри системы. В случае vk такая сущность называется «Сообщество». Это не обязательно именно программа, разбирать сообщения в сообществе могут и реальные люди, но именно разделение на пользователей и сообщества позволяет ввести особые возможности по автоматизации деятельности последних.

В описании API обязательно присутствует информация о том, кто может вызывать тот или иной сервис: пользователь и/или бот и/или кто-то еще. Для vk.com данная информация выглядит так:

Users > users.get

Возвращает расширенную информацию о пользователях.



Этот метод можно вызвать с [сервисным ключом доступа](#). Возвращаются только общедоступные данные.



Этот метод можно вызвать с [ключом доступа пользователя](#).



Этот метод можно вызвать с [ключом доступа сообщества](#).

Мы рассмотрим несколько примеров как разрабатывать программы на Python, которые действуют как от имени пользователя, так и от имени сообщества.

Никто не запрещает нам обращаться к API vk.com с использованием методов библиотеки requests, но зачастую в PyPi уже есть библиотеки, которые немного облегчают нам доступ к API популярных сервисов. Мы рассмотрим библиотеку vk_api: одну из самых популярных, но не единственную. Сначала ее надо установить:

```
pip install vk_api
```

Выполнение запросов от имени пользователя

Для выполнения запросов от имени пользователя необходимо указать его логин и пароль в своей программе (или ввести с клавиатуры, считать из файла).

У vk.com есть достаточно подробное описание сервисов, доступное [тут](#). Модуль vk_api представляет из себя некоторую обертку, которая делает часть работы по формированию запроса и разбору ответа за нас. Давайте получим три сообщения со стены пользователя, начиная со второго, то есть обратимся к методу [wall.get](#) API.

```
import vk_api

def main():
    login, password = LOGIN, PASSWORD
    vk_session = vk_api.VkApi(login, password)
    try:
        vk_session.auth(token_only=True)
    except vk_api.AuthError as error_msg:
```

```

        print(error_msg)

    return

vk = vk_session.get_api()

# Используем метод wall.get

response = vk.wall.get(count=3, offset=1)

if response['items']:

    for i in response['items']:

        print(i)

if __name__ == '__main__':

    main()

```

Для доступа к API необходимо создать объект типа `VkApi`, который принимает на вход два обязательных параметра — логин и пароль пользователя. После этого пытаемся авторизоваться, если нам это не удастся, печатаем текст ошибки и завершаем работу программы. После успешной авторизации необходимо создать объект класса `VkApiMethod`, который позволит обращаться к методам API как к методам класса. То есть для обращения к сервису `wall.get` API необходимо написать код `vk.wall.get()`, где `vk`: `VkApiMethod`. В качестве аргументов метода надо передать параметры вызова сервиса.

Обратите внимание: все методы, которые отвечают за вызов сервисов API, не содержат позиционных аргументов, только именованные.

При вызове сервиса получения сообщений со стены указываем параметр `count` — количество записей, затем `offset` — необязательный параметр, который указывает на то, начиная с какой записи надо возвращать результат. При вызове метода можно также указать `owner_id` — идентификатор пользователя или сообщества, со стены которых надо вернуть записи. По умолчанию записи возвращаются для пользователя, чей логин и пароль мы ввели. Результат нам вернется в виде json, описание полей которого можно посмотреть в документации.

Проблема с авторизацией может возникнуть не только в случае неправильного логина или пароля. Приведенный пример не сработает, если у пользователя, под которым мы пытаемся авторизоваться, настроена двухфакторная аутентификация. В этом случае пользователю нужно дополнительно ввести одноразовый код для подтверждения полномочий нашей программы. Чтобы корректно обрабатывать двухфакторную аутентификацию, необходимо написать такой код:

```
import vk_api

def auth_handler():

    """ При двухфакторной аутентификации вызывается эта
    функция. """

    # Код двухфакторной аутентификации,
    # который присылается по смс или уведомлением в
    # мобильное приложение

    key = input("Enter authentication code: ")

    # Если: True - сохранить, False - не сохранять.

    remember_device = True

    return key, remember_device

def main():

    login, password = LOGIN, PASSWORD

    vk_session = vk_api.VkApi(

        login, password,

        # функция для обработки двухфакторной
        # аутентификации

        auth_handler=auth_handler

    )

    try:

        vk_session.auth(token_only=True)

    except vk_api.AuthError as error_msg:
```

```

        print(error_msg)

    return

# ...

if __name__ == '__main__':
    main()

```

Рассмотрим еще несколько примеров обращения к API. Получим список своих друзей, то есть вызовем метод API [friends.get](#). Помимо основных полей, которые возвращает сервис, попросим еще дату рождения и город:

```

response = vk.friends.get(fields="bdate, city")

if response['items']:
    for i in response['items']:
        print(i)

```

Получим информацию о каком-нибудь пользователе по id — [users.get](#):

```

response =
vk.users.get(user_id=идентификатор_пользователя)

print(response)

```

Загрузка файлов

API vk.com поддерживает загрузку разных типов файлов, причем для каждого из типов — изображения, видео, документы — используются разные сервисы. Однако вне зависимости от типа файла весь процесс сводится к тому, что надо загрузить файл на сервер и получить его идентификатор, а затем прикладывать идентификатор туда, куда мы хотим приложить изображение: в сообщение, на стену, в ленту сообщества и т. д. Подробное описание можно прочитать [тут](#). В модуле vk_api все методы по загрузке файлов обернуты в класс VkUpload.

Рассмотрим, как это работает, на примере загрузки изображения и создания сообщения с ним на стену:

```

upload = vk_api.VkUpload(vk_session)

photo = upload.photo_wall(['Picture1.png']
)

```

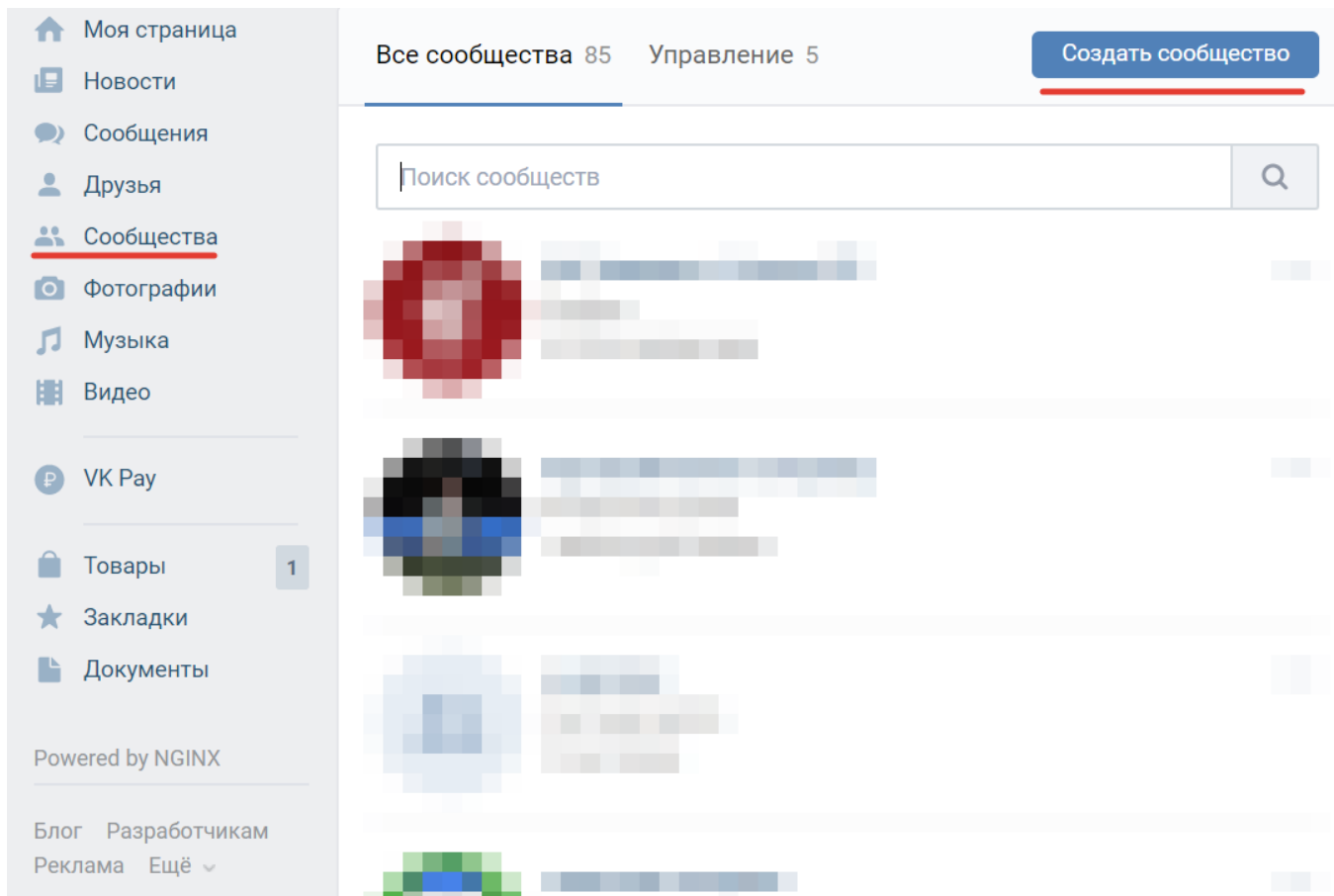
```
vk_photo_id =  
f"photo{photo[0]['owner_id']}_{photo[0]['id']}"  
  
print(photo, vk_photo_id, sep="\n")  
  
vk = vk_session.get_api()  
  
vk.wall.post(message="Test", attachments=[vk_photo_id])
```

Для загрузки фотографии для стены создадим объект типа `VkUpload`, и потом вызовем у него метод `photo_wall`, который принимает список имен файлов, которые мы хотим загрузить (грузить можно сразу несколько штук), после чего получаем идентификатор изображения вида `<type><owner_id>_<media_id>`, где тип — `photo`, а оставшиеся части можно получить из результата загрузки изображения. Вызываем сервис [wall.post](#) с помощью метода `wall.post` объекта типа `VkApiMethod`, где в качестве параметра указываем сообщение `message` и список идентификаторов вложений `attachments`.

Работа с событиями и чатами

К сожалению, в настоящее время работа с событиями чатов для пользовательских программ ограничена из соображения безопасности (вам могут сделать доступ, если вы свяжетесь с `vk.com` и докажете, что делаете не мошенническую программу, а, например, альтернативный клиент сообщений), поэтому для работы с чатами нам надо будет создать сообщество, но большая часть событий, не связанных с чатами, работает и для клиентских программ.

Создадим сообщество.



После этого перейдите в управление сообществом в раздел **Работа с API**.

Включите там LongPoll API, укажите версию API (примеры ниже тестировались на версии API 5.103), выберите типы событий, оповещения о которых вы хотите получать, а затем создайте ключ доступа.

Ключи доступа 1

Callback API

Long Poll API

Настройки

Типы событий

Long Poll API: Включен

Версия API: 5.103




Long Poll API позволяет работать с событиями из Вашего сообщества в реальном времени.

Вы можете получать обновления с помощью запросов к специальному URL. В отличие от Callback API, в этом случае мы не будем присылать отдельное уведомление на Ваш сервер для каждого события.

Как работать с Long Poll API


Также в настройках в пункте «Сообщения» включите эту функцию.

Кроме того, не забудьте на главной странице управления сообществом разрешить сообщения.

 Написать сообщение


 Управление


 Сообщения


 Статистика

 Комментарии


 События

 Реклама сообщества

 Включить уведомления

 Пригласить друзей

 Разрешить сообщения

 Добавить в левое меню

 Сохранить в закладках

 Перевести в страницу

Подготовительные работы готовы. Теперь напомним такую программу:

```
import vk_api

from vk_api.bot_longpoll import VkBotLongPoll,
VkBotEventType

import random

def main():

    vk_session = vk_api.VkApi(
```

```

token=TOKEN)

longpoll = VkBotLongPoll(vk_session, id_сообщества)

for event in longpoll.listen():

    if event.type == VkBotEventType.MESSAGE_NEW:
        print(event)
        print('Новое сообщение:')
        print('Для меня от:',
event.obj.message['from_id'])
        print('Текст:', event.obj.message['text'])
        vk = vk_session.get_api()

vk.messages.send(user_id=event.obj.message['from_id'],
                  message="Спасибо, что
написали нам. Мы обязательно ответим",
                  random_id=random.randint(0,
2 ** 64))

if __name__ == '__main__':
    main()

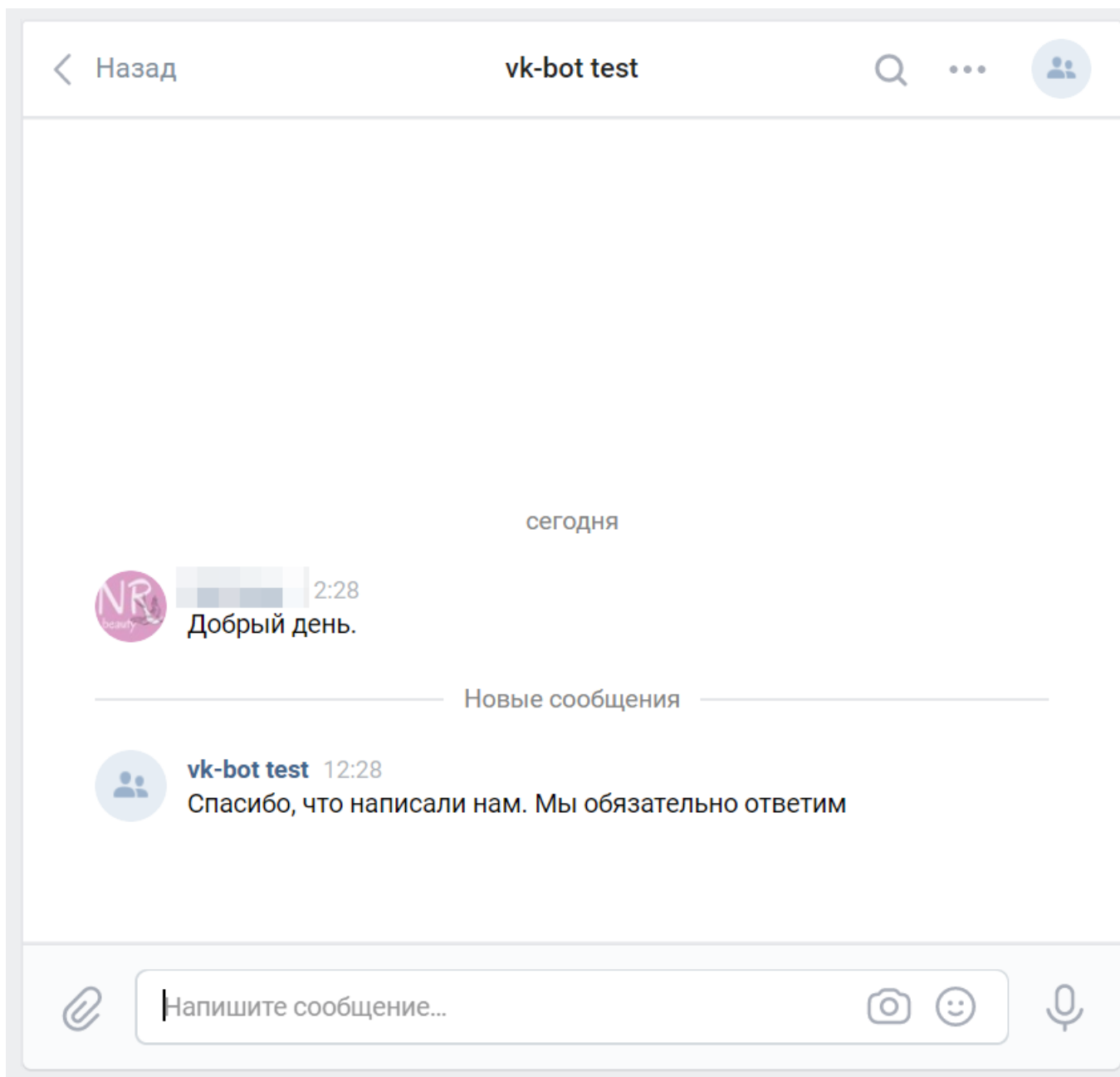
```

В этот раз при создании `VkApi` мы используем не логин/пароль, а ключ доступа. После этого создаем очередь сообщений, которую мы хотим прослушивать. Для сообществ надо использовать класс `VkBotLongPoll`, который принимает сессию подключения и идентификатор сообщества (его можно посмотреть в ссылке на страницу сообщества — это только цифры). Если мы действуем не как сообщество, а как пользователь, надо использовать класс `VkLongPoll`. После чего запускаем бесконечный цикл ожидания сообщений в нашей очереди.

Если приходит сообщение типа `VkBotEventType.MESSAGE_NEW`, мы выводим само сообщение (там много интересной информации), затем — `id` пользователя-автора сообщения и текст сообщения. Кроме

того, мы можем сразу ответить пользователю, вызвав сервис [messages.send](#).

Для корректного вызова необходимо передать `user_id` — идентификатор пользователя или группового чата, куда мы пишем сообщение, текст сообщения `message`, а также большое случайное число `random_id`, которое нужно для того, чтобы не отправлять пользователю одни и те же сообщения несколько раз.



В настоящий момент `vk_apr` поддерживает следующие типы событий:

```
MESSAGE_NEW = 'message_new'
MESSAGE_REPLY = 'message_reply'
MESSAGE_EDIT = 'message_edit'
```

```
MESSAGE_TYPING_STATE = 'message_typing_state'
MESSAGE_ALLOW = 'message_allow'
MESSAGE_DENY = 'message_deny'
PHOTO_NEW = 'photo_new'
PHOTO_COMMENT_NEW = 'photo_comment_new'
PHOTO_COMMENT_EDIT = 'photo_comment_edit'
PHOTO_COMMENT_RESTORE = 'photo_comment_restore'
PHOTO_COMMENT_DELETE = 'photo_comment_delete'
AUDIO_NEW = 'audio_new'
VIDEO_NEW = 'video_new'
VIDEO_COMMENT_NEW = 'video_comment_new'
VIDEO_COMMENT_EDIT = 'video_comment_edit'
VIDEO_COMMENT_RESTORE = 'video_comment_restore'
VIDEO_COMMENT_DELETE = 'video_comment_delete'
WALL_POST_NEW = 'wall_post_new'
WALL_REPOST = 'wall_repost'
WALL_REPLY_NEW = 'wall_reply_new'
WALL_REPLY_EDIT = 'wall_reply_edit'
WALL_REPLY_RESTORE = 'wall_reply_restore'
WALL_REPLY_DELETE = 'wall_reply_delete'
BOARD_POST_NEW = 'board_post_new'
BOARD_POST_EDIT = 'board_post_edit'
BOARD_POST_RESTORE = 'board_post_restore'
BOARD_POST_DELETE = 'board_post_delete'
MARKET_COMMENT_NEW = 'market_comment_new'
MARKET_COMMENT_EDIT = 'market_comment_edit'
MARKET_COMMENT_RESTORE = 'market_comment_restore'
MARKET_COMMENT_DELETE = 'market_comment_delete'
```

```
GROUP_LEAVE = 'group_leave'

GROUP_JOIN = 'group_join'

USER_BLOCK = 'user_block'

USER_UNBLOCK = 'user_unblock'

POLL_VOTE_NEW = 'poll_vote_new'

GROUP_OFFICERS_EDIT = 'group_officers_edit'

GROUP_CHANGE_SETTINGS = 'group_change_settings'

GROUP_CHANGE_PHOTO = 'group_change_photo'

VKPAY_TRANSACTION = 'vkpay_transaction'
```

Например, так можно узнать, когда кто-то начал писать сообществу сообщение в чат:

```
if event.type == VkBotEventType.MESSAGE_TYPING_STATE:

    print(f'Печатает {event.obj.from_id} для
{event.obj.to_id}')
```

А так, что пользователь добавился в сообщество:

```
if event.type == VkBotEventType.GROUP_JOIN:

    print(f'{event.obj.user_id} вступил в группу!')
```

Поэкспериментируйте с сообществом с запущенной программой обработки события, выводите их через `print(event.type)`, посмотрите, в каком случае возникают события того или иного вида (об этом достаточно просто догадаться по названию события).

Заключение

API `vk.com` очень обширно и мы лишь немного прикоснулись к нему. Однако надеемся, что это даст вам импульс к самостоятельному изучению и написанию приложений для этой платформы.

Запрос Записи на стене

Создайте запрос, который получает для последних пяти записей на стене их текст и время создания. Это время передается в формате `unixtime`. Используя библиотеку `datetime`, переведите ее в «человеческий» формат. Выведите результат в формате:

```
{text_of_the_note};
date: {2020-02-02}, time: {20:20:20}
```

Во всех задачах этой темы при сдаче задач на проверку замените строку с вашими логином и паролем на такую:

login, password = LOGIN, PASSWORD

Запрос Сортированные друзья

Создайте запрос, который получает и выводит друзей пользователя, отсортированных по фамилии по алфавиту. Каждого друга вывести с новой строки и указать для него фамилию, имя, дату рождения.

Загрузка файлов

Напишите программу, которая загружает сразу несколько файлов в основной альбом фотографий сообщества. Для этого создайте в папке с программой папку `static`, а в ней — папку `img`, куда и положите картинки или фотографии для загрузки. Для загрузки фото в конкретный альбом можно воспользоваться методом

```
upload.photo(filename, album_id, group_id)
```

Файл с программой загрузите в качестве решения.

Бот Большой Брат

Напишите бота, который в ответ на сообщение пользователя

пишет: "Привет, {имя}!"

Имя нужно определить из события получения нового сообщения (`message['from_id']`).

По `id` получаем сведения о пользователе: имя и город (если указан). Если есть город, то нужно в сообщении написать еще и "Как поживает {город}?"

Бот Дата-Время

Если в новом сообщении пользователя есть слова: «время», «число», «дата», «день», нужно сообщить ему сегодняшнюю дату, московское время и день недели. Если нет, то сообщить о такой возможности.

Используйте библиотеку `datetime`.

Вики-бот

Напишите бота, который в ответ на сообщение пользователя спрашивает, что хочет узнать пользователь, а затем присылает требуемую информацию. Для взаимодействия с Википедией можно установить модуль `wikipedia`.

Документацию по API Википедии можно посмотреть [здесь](#). Добавьте возможность после ответа задать следующий вопрос.

Получение фото из альбома

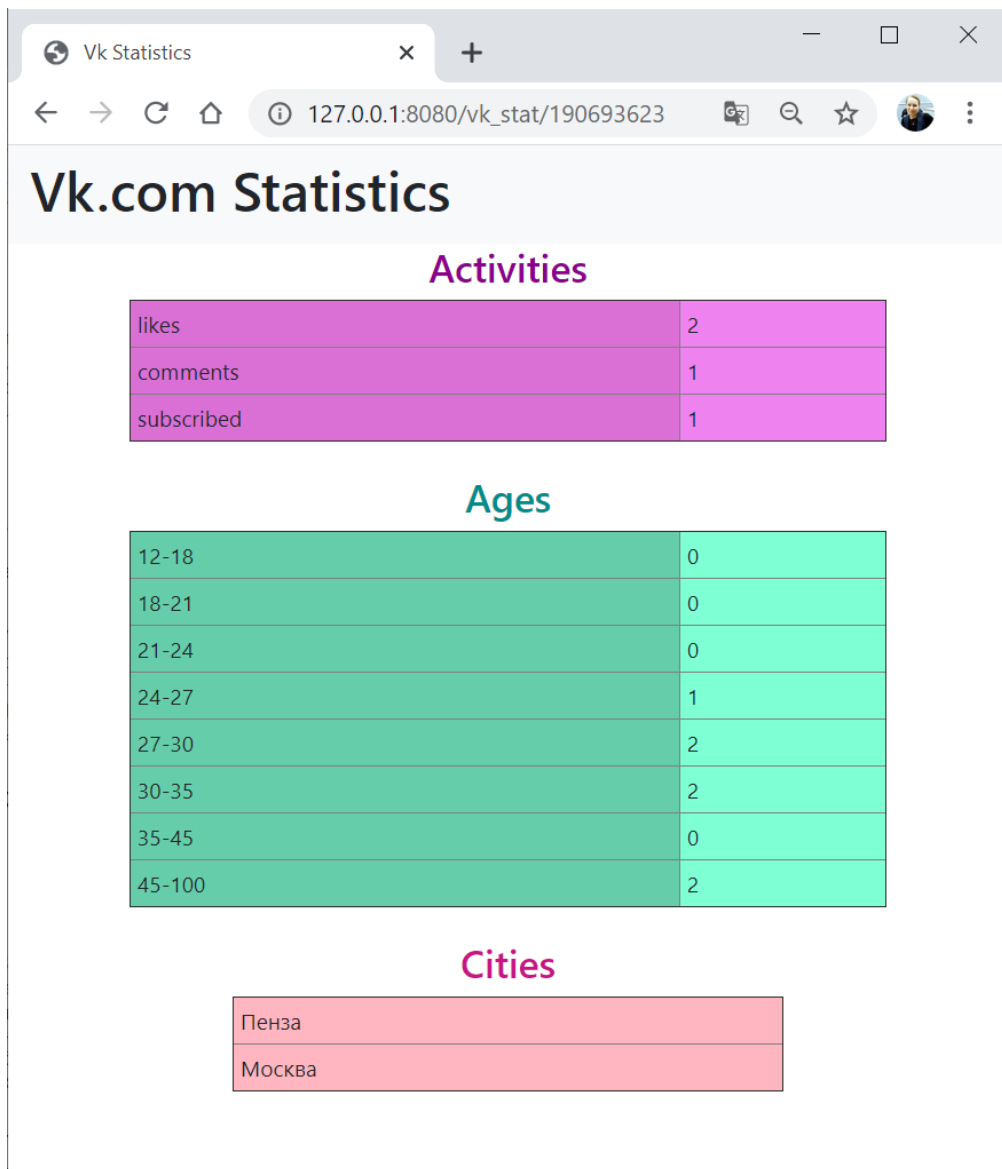
Предположим, мы хотим разместить где-либо ссылки на фотографии из альбома сообщества. Для этого нужно написать программу, которая с помощью метода `photos.get(album_id, group_id)` получит и распечатает (пока что) url и размеры всех фотографий из данного альбома.

Бот с картинкойЗапрос Статистика

Создайте запрос, который получает статистику по группе и выводит ее в виде таблицы в браузере по адресу `/vk_stat/<int:group_id>`

Для подсчета статистики используйте метод `stats.get(group_id, fields=reach)`. Получите последних 10 периодов получения статистики. Для временных периодов, в которых была активность, подсчитайте, например, сколько было получено комментариев, лайков и ответов; как распределена активность участников группы по возрастным категориям; из каких они городов (без повторений).

Используйте flask. Архив с сервером и html шаблоном загрузите в качестве ответа.



В ответ на сообщение пользователя бот здоровается с ним по имени и посылает случайную картинку из альбома сообщества.

Нам нужно получить `photo_id`, чтобы прикрепить его к сообщению. Для этого необходимо воспользоваться решением предыдущей задачи: авторизоваться как пользователю и получить список `id` картинок из альбома группы в формате `photo{owner_id}_{photo_id}`.

Бот дня недели

В ответ на новое сообщение пользователя бот пишет, что может сказать, в какой день недели была какая-нибудь дата и просит пользователя ввести ее в формате **YYYY-MM-DD**. И возвращает ответ. И можно спрашивать бесконечно!

Бот-геокодер с клавиатурой

В ответ на новое сообщение пользователя бот предлагает ввести название местности, которую хочет увидеть пользователь. После получения названия, бот показывает клавиатуру и предлагает выбрать тип карты. Показывает карту с подписью в виде Это {местность}. Что вы еще хотите увидеть? И ждет следующего запроса.