

Создание и запуск чат-бота

Методические рекомендации
к занятию 3

Сервисы Telegram, GigaChat и другие упоминаются только в образовательных целях и не являются рекламой.

Создание бота с GigaChat API (90 мин.)

Цели занятия

1. Повторить и закрепить базовые навыки разработки Telegram-ботов.
2. Научиться работать с сервисом GigaChat API: получить токен, отправить запрос на генерацию текста и обработать ответ.
3. Собрать итоговый проект — «умного» Telegram-бота, способного принимать сообщения от пользователя и отвечать в научном стиле, используя модель GigaChat API.

Структура урока

1. Вступление и мотивация (10 мин.).
2. Обзор GigaChat и подготовка окружения (20 мин.).
3. Разбор кода и объяснение ключевых блоков (35 мин.).
4. Практика и тестирование (15 мин.).
5. Ответы на вопросы, итоги занятия и домашнее задание (10 мин.).

Общее время: 90 мин.

1. Вступление и мотивация (10 мин.)

Цели

- Напомнить обучающимся, что было сделано на предыдущих уроках (базовые Telegram-боты, простая работа с внешними API).

- Рассказать об особенностях новой задачи: интегрировать в свой чат-бот полноценную языковую модель (GigaChat), которая будет генерировать ответы на русском языке.

Шаги

1. Повторение

- Спросите у обучающихся, что они помнят из первых двух занятий:
 - Как запускать бота (функция `main()`, `ApplicationBuilder().token(...).build()`).
 - Как работать с обработчиком сообщений: `MessageHandler(filters.TEXT, handle_message)`.
- Уточните, сталкивался ли кто-то уже с внешними API (например, на прошлом уроке подключали другую службу). Кратко расскажите, что такое API.

2. Мотивация

- Объясните, что теперь бот станет «умнее» за счёт подключённой языковой модели: он сможет отвечать на вопросы не заготовленными фразами, а формировать осмысленные ответы в научном стиле.
- Расскажите о том, где это востребовано (сервисные чат-боты, ассистенты, справочные системы).

2. Обзор GigaChat и подготовка окружения (20 мин.)

Цели

- Показать, как устроен GigaChat и как в нём авторизоваться.
- Изучить, какие методы API для запросов информации с сервера могут понадобиться.

Шаги

2.1. Что такое GigaChat

- Кратко объясните, что GigaChat — это сервис (модель), предоставляющий API для генерации текста.
- GigaChat — это аналог других языковых моделей (GPT), но с собственными особенностями.

2.2. Регистрируемся и получаем ключи

- Покажите, что для интеграции с GigaChat понадобятся три значения:
 - `GIGACHAT_CLIENT_ID` (идентификатор клиента);
 - `GIGACHAT_CLIENT_SECRET` (секретный ключ в Base64 — бинарном формате хранения);
 - `GIGACHAT_SCOPE` (зона доступа внутри API, например `GIGACHAT_API_PERS`).

- Подчеркните, что в реальном проекте эти ключи нельзя выкладывать в открытый доступ. Во время обучения используются тестовые или пробные ключи.

2.3. Окружение

1. Python и библиотеки

На предыдущих занятиях обучающиеся уже работали с `python-telegram-bot`.

Напомните, что нужно установить `requests` (если ещё не установили):

bash

```
pip install requests
```

2. SSL-сертификаты

Обратите внимание на то, что в коде, который понадобится нам сегодня, используется `verify=False` для упрощения. Это отключает проверку SSL-сертификата. Предупредите, что в реальных проектах так делать небезопасно: подключение без подписанных протоколов в некоторых случаях может привести к утечке личной информации.

3. Разбор кода и объяснение ключевых блоков (35 мин.)

Цель

Понять и дописать/исправить код.

Теперь переходим к коду, который обучающиеся должны будут понять и дописать/исправить при необходимости, чтобы бот заработал. Ниже — те же блоки, что и в конечном решении, с дополнительными пояснениями.

3.1. Импорт и переменные

```
import base64
import uuid
import requests
import json
from telegram import Update
from telegram.ext import ApplicationBuilder, MessageHandler,
ContextTypes, filters

# Замените на реальные значения
TELEGRAM_TOKEN = "Введите свой ключ"
GIGACHAT_CLIENT_ID = "Введите свой ключ"
GIGACHAT_CLIENT_SECRET = "Введите свой ключ"
GIGACHAT_SCOPE = "GIGACHAT_API_PERS" # Используйте нужное
значение scope
GIGACHAT_MODEL = "GigaChat-Max" # Модель для генерации
текста
```

1. Подключаем библиотеки

- `requests` и `json` — для работы с запросами к GigaChat;
- `uuid` — генерируем уникальные идентификаторы;
- `telegram` — из пакета `python-telegram-bot` (с помощью него будет работать бот, связанный с GigaChat);
- `base64` — возможно (если понадобится кодирование токенов).

2. Переменные

- `TELEGRAM_TOKEN`: токен телеграм-бота от `BotFather` (обучающиеся помнят с прошлых занятий);
- `GIGACHAT_CLIENT_ID`, `GIGACHAT_CLIENT_SECRET`: ключи от GigaChat;
- `GIGACHAT_SCOPE`: область доступа (чаще всего `GIGACHAT_API_PERS`);
- `GIGACHAT_MODEL`: указываем, какую конкретно модель мы хотим использовать (зависит от настроек GigaChat).

3.2. Функция `get_gigachat_token()`

```
def get_gigachat_token() -> str:
    """
    Получаем токен доступа:
    URL: https://ngw.devices.sberbank.ru:9443/api/v2/oauth
    Payload: scope=<SCOPE>
    Заголовки: Content-Type, Accept, RqUID, Authorization (Basic
    <TOKEN>).
    """
    url = "https://ngw.devices.sberbank.ru:9443/api/v2/oauth"
    payload = f"scope={GIGACHAT_SCOPE}"
    rq_uid = str(uuid.uuid4())

    headers = {
        "Content-Type": "application/x-www-form-urlencoded",
        "Accept": "application/json",
        "RqUID": rq_uid,
        "Authorization": f"Basic {GIGACHAT_CLIENT_SECRET}"
    }

    # Отключаем проверку SSL-сертификатов (только для
    разработки!)
    response = requests.post(url, headers=headers, data=payload,
    verify=False)
    if response.status_code == 200:
        token = response.json().get("access_token")
        return token
    else:
        print("Ошибка получения токена:", response.status_code,
        response.text)
        return ""
```

1. Назначение

- Из примера видно, что нужно сформировать POST-запрос (запрос на сервер с целью отправки информации) на <https://ngw.devices.sberbank.ru:9443/api/v2/oauth>, передать `scope`, уникальный идентификатор запроса (`RqUID`) и авторизоваться в заголовке `Authorization` (в формате `Basic <...>`).
- Сервис возвращает JSON (файловый формат со словарной структурой), где хранится `access_token`. Его и нужно вытащить.

2. Обратите внимание

- `verify=False` — режим без проверки сертификата. В реальном проекте так не стоит делать. Это может привести к гипотетической утечке данных.
- Если статус-код не `200` (`http response 200`, говорящий о том, что всё идёт корректно), функция вернёт пустую строку `""`, и бот поймёт, что не смог авторизоваться.

3.3. Функция-обработчик `handle_message()`

```
async def handle_message(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    """
    Обработка входящего сообщения пользователя и отправляем
    его в GigaChat API для генерации ответа.
    ...
    """
    user_text = update.message.text

    token = get_gigachat_token()
    if not token:
        await update.message.reply_text("Ошибка авторизации
в GigaChat API.")
        return

    url = "https://gigachat.devices.sberbank.ru/api/v1/chat/
completions"
    payload = json.dumps({
        "model": GIGACHAT_MODEL,
        "messages": [
            {
                "role": "system",
                "content": "Отвечай как научный сотрудник"
            },
            {
                "role": "user",
                "content": user_text
            }
        ]
    })
```

```
        }
    ],
    "profanity_check": True
})
headers = {
    "Content-Type": "application/json",
    "Accept": "application/json",
    "Authorization": f"Bearer {token}"
}
response = requests.post(url, headers=headers, data=payload,
verify=False)
if response.status_code == 200:
    resp_json = response.json()
    # Предполагаем, что ответ модели находится в первом
    элементе массива choices
    answer = resp_json.get("choices", [{}])[0].get("message",
    {}).get("content", "")
    if answer:
        await update.message.reply_text(answer)
    else:
        await update.message.reply_text("Не удалось
получить корректный ответ от GigaChat API.")
    else:
        error_msg = f"Ошибка при получении ответа
от GigaChat API: {response.status_code}"
        print(error_msg, response.text)
        await update.message.reply_text(error_msg)
```

1. Суть

Когда пользователь присылает текст, мы:

- Получаем свежий токен через `get_gigachat_token()`.
- Формируем запрос для GigaChat (`url`, `payload`, `headers`).
- Отправляем текст (`user_text`) и фиксируем желаемый «тон» в `messages[0]` (`system`).
- Получаем ответ от модели и отправляем пользователю через `update.message.reply_text(...)`.

2. Логика с `role`

- `system` — это инструкция для модели, как вести себя в диалоге;
- `user` — это реплика пользователя.

Можно расширять массив для более сложного контекста (например, запоминать предыдущие сообщения).

3.4. Функция `main()`

```
def main() -> None:
    """Запуск Telegram-бота."""
    application = ApplicationBuilder().token(TELEGRAM_TOKEN).
    build()
    application.add_handler(MessageHandler(filters.TEXT &
    (~filters.COMMAND), handle_message))
    print("Бот запущен...")
    application.run_polling()
```

1. Принцип работы

- Создаём бота через `ApplicationBuilder().token(...)`.
- Регистрируем `MessageHandler`, который будет принимать все текстовые сообщения (кроме команд) и обрабатывать их функцией `handle_message()`.
- Запускаем бесконечный цикл `run_polling()`.

2. Вывод отладки

`print("Бот запущен...")` позволяет понять, что скрипт стартовал успешно.

3.5. Запуск

python

```
if __name__ == '__main__':
    main()
```

Всё стандартно: вызываем `main()`, запуская бота.

4. Практика и тестирование (15 мин.)

Цель

Убедиться, что у каждого обучающегося всё работает, токен корректно используется, ответы от GigaChat приходят.

Рекомендации

1. Проверка среды

- Убедитесь, что обучающиеся импортировали все нужные библиотеки (`requests`, `python-telegram-bot`, `uuid`, `base64` и т. д.).
- На локальных машинах может потребоваться отключить проверку SSL-сертификатов, что в коде уже учтено флагом `verify=False`.

2. Локальный запуск

- Запускаем скрипт командой:

```
bash  
python bot.py
```
- Убеждаемся, что в консоли появляется «Бот запущен...».

3. Тест в Telegram

- Заходим в Telegram, находим созданного ранее бота (имя/юзернейм).
- Пишем ему произвольное сообщение.
- Если всё в порядке, бот отвечает текстом в научном стиле.
- Если токен неверный или закончился, бот выдаст «Ошибка авторизации в GigaChat API».

4. Частые проблемы

- **Неверный TELEGRAM_TOKEN.** Бот не запустится или не будет отвечать. Решение: скопировать верный токен.
- **Старые/неправильные ключи GIGACHAT.** `get_gigachat_token()` вернёт пустой ответ. Решение: обновить ключ.
- **Ошибка SSL.** Если у кого-то на компьютере установлены политики безопасности и не удаётся отключить SSL-проверку. Решение: заменить компьютер на другой, с открытыми политиками.

5. Ответы на вопросы, итоги занятия и домашнее задание (10 мин.)

Ответы на вопросы

Позвольте обучающимся задать все возникшие вопросы.

Итоги занятия

1. Полученные навыки

- Обучающиеся поняли, как использовать `requests` для авторизации (OAuth) и для генерации текста через GigaChat API.
- Освоили использование `MessageHandler` без команд для обработки любого текста.
- Научились моделировать тон ответов, задавая `role = "system"`.

2. Домашнее задание (опционально)

- **Задание 1.** Добавить к массиву `messages` несколько предыдущих вопросов и ответов, чтобы бот «помнил» часть диалога.
- **Задание 2.** Сделать так, чтобы бот менял стиль ответа с помощью `/mode`. Например, `/mode fairy` (сказочный персонаж) или `/mode professor` (научный сотрудник).

- **Дополнительное задание.** Включить `verify=True` и разобраться, как добавить или указать сертификаты, чтобы не отключать SSL-проверку.

3. Подводим итог

- Обучающиеся создали Telegram-бота, который умеет говорить с ними на русском языке в научном стиле, при этом отвечает действительно развёрнуто благодаря GigaChat.
- Следующим шагом может быть расширение функционала: сохранение истории общения, фильтрация пользователей, написание тестов или деплой (публикация бота на сервере с дальнейшей его настройкой и запуском) бота на удалённый сервер.

```
import base64
import uuid
import requests
import json
from telegram import Update
from telegram.ext import ApplicationBuilder, MessageHandler,
ContextTypes, filters

# Замените на реальные значения
TELEGRAM_TOKEN = "Введите свой ключ"
GIGACHAT_CLIENT_ID = "Введите свой ключ"
GIGACHAT_CLIENT_SECRET = "Введите свой ключ"
GIGACHAT_SCOPE = "GIGACHAT_API_PERS" # Используйте нужное
значение scope
GIGACHAT_MODEL = "GigaChat-Max" # Модель для генерации текста

def get_gigachat_token() -> str:
    """
    Получаем токен доступа по примеру Postman:
    URL: https://ngw.devices.sberbank.ru:9443/api/v2/oauth
    Payload: scope=<SCOPE>
    Заголовки: Content-Type, Accept, RqUID, Authorization
    (Basic <TOKEN>).
    """
    url = "https://ngw.devices.sberbank.ru:9443/api/v2/oauth"
    payload = f"scope={GIGACHAT_SCOPE}"
    rq_uid = str(uuid.uuid4())
```

```
headers = {
    "Content-Type": "application/x-www-form-urlencoded",
    "Accept": "application/json",
    "RqUID": rq_uid,
    "Authorization": f"Basic {GIGACHAT_CLIENT_SECRET}"
}
# Отключаем проверку SSL-сертификатов (только для
разработки!)
response = requests.post(url, headers=headers, data=payload,
verify=False)
if response.status_code == 200:
    token = response.json().get("access_token")
    return token
else:
    print("Ошибка получения токена:", response.status_code,
response.text)
    return ""

async def handle_message(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    """
    Обработываем входящее сообщение пользователя и отправляем
его в GigaChat API для генерации ответа.
    Пример генерации текста по Postman:
        URL: https://gigachat.devices.sberbank.ru/api/v1/chat/
completions
        Заголовки: Content-Type, Accept, Authorization: Bearer
<TOKEN>
        Тело запроса включает модель, список сообщений и параметр
profanity_check.
    """
    user_text = update.message.text

    token = get_gigachat_token()
    if not token:
        await update.message.reply_text("Ошибка авторизации
в GigaChat API.")
    return
```

```
url = "https://gigachat.devices.sberbank.ru/api/v1/chat/
completions"
payload = json.dumps({
    "model": GIGACHAT_MODEL,
    "messages": [
        {
            "role": "system",
            "content": "Отвечай как научный сотрудник"
        },
        {
            "role": "user",
            "content": user_text
        }
    ],
    "profanity_check": True
})
headers = {
    "Content-Type": "application/json",
    "Accept": "application/json",
    "Authorization": f"Bearer {token}"
}
response = requests.post(url, headers=headers, data=payload,
verify=False)
if response.status_code == 200:
    resp_json = response.json()
    # Предполагаем, что ответ модели находится в первом
элементе массива choices
    answer = resp_json.get("choices", [{}])[0].get("message",
{}).get("content", "")
    if answer:
        await update.message.reply_text(answer)
    else:
        await update.message.reply_text("Не удалось полу-
чить корректный ответ от GigaChat API.")
else:
    error_msg = f"Ошибка при получении ответа от GigaChat
API: {response.status_code}"
    print(error_msg, response.text)
    await update.message.reply_text(error_msg)
```

```
def main() -> None:
    """Запуск Telegram-бота."""
    application = ApplicationBuilder().token(TELEGRAM_TOKEN).
    build()

    application.add_handler(MessageHandler(filters.TEXT &
    (~filters.COMMAND), handle_message))
    print("Бот запущен...")
    application.run_polling()

if __name__ == '__main__':
    main()
```