

Создание и запуск чат-бота

Конспект занятия 3

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

1. Вступление и мотивация

Цели этого этапа

- Вспомнить, что было сделано на предыдущих уроках: алгоритм создания бота, базовые функции, Reply-клавиатуры, Inline-кнопки.
- Понять, о чём будет идти речь на занятии: о подключении полноценной языковой модели GigaChat, которая будет формировать «умные» ответы на русском языке в заданном стиле.

Ключевые моменты для повторения

1. Как запускать бота

Используем функцию `main()` и конструкцию:

```
application = ApplicationBuilder().token(TELEGRAM_TOKEN).build()
application.run_polling()
```

2. Как работает обработчик сообщений

`MessageHandler(filters.TEXT, handle_message)` перехватывает любые текстовые сообщения от пользователя в боте.

Зачем нужен GigaChat

GigaChat нужен для создания помощников, справочных систем и сервисных чат-ботов, которые будут генерировать полноценные ответы на русском языке в заданном стиле общения (в нашем случае — научном).

2. Обзор GigaChat API и подготовка окружения

Цели этого этапа

- Понять, что такое GigaChat.
- Освоить процедуру регистрации и получения ключей.
- Подготовить рабочее окружение.

2.1. Немного о GigaChat

- Сервис (модель) для генерации текста — аналог GPT, но со своими особенностями.
- Предоставляет [API](#) (интернет-сервисы для взаимодействия с серверами — удалёнными компьютерами) для интеграции в проекты.
- Для работы требуется авторизация (OAuth) и отправка специальных запросов.

2.2. Регистрируемся и получаем ключи

Подробнее об этом в [инструкции](#).

Чтобы пользоваться GigaChat, нужны **три** переменные:

- **GIGACHAT_CLIENT_ID** — идентификатор клиента (ID).
- **GIGACHAT_CLIENT_SECRET** — «секрет» (в бинарном формате хранения Base64), который будет использоваться в заголовке запроса Authorization.
- **GIGACHAT_SCOPE** — область доступа, чаще всего **GIGACHAT_API_PERS**.

Никогда не выкладывайте значения этих переменных в открытый доступ!

2.3. Подготовка окружения

Устанавливаем необходимые библиотеки:

bash

pip install requests

- Проект использует **python-telegram-bot** (библиотеку для работы с ботами), **requests** (библиотеку для создания запросов в интернете), **uuid** (библиотеку для создания уникальных номеров запросов), **json** (библиотеку для работы с json-форматом хранения данных), **base64** (библиотеку для работы с данными, хранимыми в бинарном виде).
- В примере кода иногда используется **verify=False** для упрощения SSL-проверки (проверки подписанных сертификатов для безопасной работы на сайте). **В реальных проектах так не делают** — нужно настроить сертификаты и использовать **verify=True**.

3. Разбор кода и объяснение ключевых блоков

Ниже приведён код итогового бота и основные комментарии по каждому блоку.

3.1. Импорт и глобальные 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
```

- **TELEGRAM_TOKEN**: ключ бота из BotFather (как на прошлых уроках).
- **GIGACHAT_CLIENT_ID** и **GIGACHAT_CLIENT_SECRET**: используются для авторизации по OAuth.
- **GIGACHAT_SCOPE**: определяет область доступа к сервису.
- **GIGACHAT_MODEL**: указывает, какую модель GigaChat мы хотим использовать. Название зависит от доступных моделей в GigaChat.

3.2. Функция get_gigachat_token()

```
def get_gigachat_token() -> str:
    """
    Получаем токен доступа по примеру:
    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-сертификата (verify=False) — только
в учебных целях
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 ""
```

Как это работает?

1. Формируется POST-запрос на страницу авторизации в GigaChat API: <https://ngw.devices.sberbank.ru:9443/api/v2/oauth>.
2. Тело запроса (payload): `scope=...`
3. В заголовках передаём:
 - `RqUID` — уникальный идентификатор запроса (используем `uuid.uuid4()`).
 - `Authorization: Basic <GIGACHAT_CLIENT_SECRET>` (секрет хранится в Base64-формате).
4. Если запрос прошёл успешно, мы берём из ответа сервера `response` значение поля `access_token`. Это как пароль, который позволит нам подключиться к нейросети и задавать ей вопросы. Пароль будет действовать примерно 30 минут или час. Для этого мы преобразуем ответ в формат `.json` и получаем значение нужного нам поля.
5. В случае ошибки возвращаем пустую строку `""`.

Если функция вернула пустую строку, значит, получить токен не удалось.

3.3. Функция-обработчик `handle_message()`

```
async def handle_message(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    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()
        # Ответ модели ищем в resp_json["choices"][0]["message"]
        ["content"]
        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)
```

Основные шаги

1. Считываем текст пользователя: `user_text = update.message.text`.
2. Получаем «свежий» токен `token = get_gigachat_token()`.
Если `token` пуст, сразу сообщаем об ошибке.
3. Формируем запрос к GigaChat по указанной в документации ссылке:
 - `url = "https://gigachat.devices.sberbank.ru/api/v1/chat/completions"`.
 - Тело (`payload`) включает:
 - `model` (какую модель используем);
 - `messages` (диалог: роли `system` и `user`);
 - `profanity_check` (включён фильтр «неприемлемой лексики»).
 - В заголовках: `Authorization: Bearer <token>` (здесь мы передаём наш токен авторизации).
4. Отправляем запрос методом `requests.post()`.
5. Если код ответа `200`, пытаемся извлечь поле `answer` из `resp_json`.
6. Отправляем ответ бота через `update.message.reply_text(answer)`.

Обратите внимание, что роль `system` задаёт стиль ответа (здесь — «отвечай как научный сотрудник»), а роль `user` содержит сообщение пользователя.

3.4. Функция `main()`

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

- Создаём приложение с токеном Telegram.
- Регистрируем обработчик для всех текстовых сообщений.
- Запускаем бота в режиме `polling` (опрос сервера Telegram).

3.5. Запуск

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

Стандартная точка входа: при запуске скрипта бот начинает работать.

4. Практика и тестирование

Цель

Запустить бота каждого ученика, проверить корректность ответа от GigaChat.

Порядок действий

1. **Установка нужных библиотек.** `pip install requests` (если ещё не стоит), а также убедиться, что установлен `python-telegram-bot`.
2. **Проверка SSL.** В коде стоит `verify=False`, что отключает проверку сертификата. Если возникнут ошибки, попробуйте установить сертификаты Минцифры для решения проблемы.

Запуск

bash

`python bot.py`

Если в консоли видим сообщение «**Бот запущен...**», всё хорошо.

3. Тестирование

- Открываем Telegram, находим своего бота (созданного заранее через BotFather).
- Пишем любое сообщение. Бот должен отвечать в научном стиле.

4. Возможные проблемы

- **Неверный `TELEGRAM_TOKEN`** — бот не запустится или не будет отвечать в Telegram.
Решение: скопировать токен корректно.
- **Неверные данные для GigaChat (просроченные ключи и т. д.)** — тогда `get_gigachat_token()` вернёт пустую строку.
Решение: скопировать или сгенерировать новые ключи.
- **Проблемы с SSL** — на ПК запрещено отключать проверку сертификатов.
Решение: сменить компьютер.

5. Часто возникающие вопросы и ответы на них

Как поменять стиль ответа?

В `messages` меняем содержимое `role="system"`, например:

json

```
{
  "role": "system",
  "content": "Отвечай как сказочный персонаж"
}
```

Как сделать бота «запоминающим» контекст диалога?

Нужно сохранять историю сообщений в отдельном массиве и при каждом запросе передавать все прошлые реплики в `messages`.

6. Домашнее задание (опционально)

Задание 1. Дополнить массив `messages`, чтобы бот «помнил» хотя бы последние 2–3 сообщения.

Задание 2. Реализовать переключение стиля ответа, например:

- `/mode fairy` → «сказочный персонаж»;
- `/mode professor` → «научный сотрудник».

7. Итоги занятия

- Повторили основы создания и настройки Telegram-бота (обработчики, запуск, тестирование).
- Разобрались, как работает авторизация (OAuth) в GigaChat, научились получать `access_token`.
- Научились взаимодействовать с GigaChat через `requests.post(...)`.

Заключение

На этом уроке мы:

1. **Повторили** базовые навыки создания Telegram-бота.
2. **Научились** получать OAuth-токен из GigaChat и отправлять запросы для генерации текста.

3. **Собрали** «умного» бота, который отвечает в научном стиле благодаря модели GigaChat.
4. **Поняли**, как можно расширять функционал: хранить контекст, менять стили, работать безопаснее с SSL и т. д.

Этот опыт показывает, как просто использовать мощную языковую модель в ботах, чтобы делать ответы более развёрнутыми и естественными.

В дальнейшем можно добавить:

- управление ролями пользователей;
- ведение истории всего диалога;
- кнопки, меню и т. д.;
- деплой (публикацию бота на сервере с дальнейшей его настройкой и запуском) на удалённый сервер, чтобы бот был доступен 24/7.

Итоговая схема взаимодействия

Давайте подытожим схему взаимодействия, которую мы изучили сегодня.

1. Пользователь → Telegram-бот
 - Пользователь в Telegram отправляет сообщение.
 - Telegram-сервер пересылает это сообщение вашему боту (через *polling* или *webhook*).
2. Telegram-бот → GigaChat API
 - Бот, получив сообщение, отправляет запрос к GigaChat API.
 - Запрос формируется (в формате JSON) и передаётся через HTTP (или HTTPS) на адрес GigaChat API вместе с нужными параметрами (текстом сообщения, настройками генерации ответа и т. д.).
3. GigaChat API → Telegram-бот
 - GigaChat API принимает запрос, обрабатывает его и формирует ответ с помощью нейросети.
 - Ответ возвращается боту также через HTTP/HTTPS, как JSON.
4. Telegram-бот → Пользователь
 - Бот получает ответ от API и «превращает» его в сообщение для Telegram (текст, фото и т. д.).
 - Отправляет ответ пользователю через метод бота **reply_text**.

Таким образом, каждый новый запрос пользователя повторяет цикл: **Сообщение от пользователя → Обработка ботом → Запрос к API → Ответ от API → Обработка ботом → Сообщение пользователю**.