Создание телеграм-бота

В этом материале мы поговорим о том, как создать собственного бота. Для этого вам понадобится:

1. получить токен для бота;
2. подключить библиотеку pyTelegramBotAPI;
3. создать архитектуру телеграм-бота;
4. настроить диалог бота с пользователем;
5. настроить работу с кнопками для прохождения сценария.

Выполним эти действия по порядку.

Получение токена

Прежде чем приступить к созданию бота, необходимо получить токен — секретный ключ, который оживит нашего виртуального помощника.

Для этого перейдите к отцу ботов BotFather и нажмите на START или «ЗАПУСТИТЬ».

Затем следуйте инструкции.

1. В появившемся сообщении со списком команд выберите **/newbot**.
2. Выберите имя для бота. При необходимости это имя можно будет потом изменить.
3. Задайте боту username — это уникальное имя, которое уже нельзя будет изменить. Оно обязательно должно оканчиваться на слово bot. Например, MyFirstTestSmartBot.
4. После выбора имени вы получите токен бота, необходимый для дальнейшей работы.
5. Теперь у вас есть секретный ключ, и вы можете переходить к следующему шагу — к подключению библиотеки.

Библиотека pyTelegramBotAPI

Есть большое количество библиотек для разработки, но мы остановимся на pyTelegramBotAPI из-за её преимуществ — простоты в использовании и большого сообщества. Она упрощает отправку и получение запросов, что позволяет сосредоточиться на разработке логики бота.

Установить библиотеку очень легко. Нужно выполнить следующую команду:

```
pip install pyTelegramBotAPI
```

Простейший бот выглядит таким образом:

```
import telebot

bot = telebot.TeleBot("") # Токен, полученный от BotFather.

@bot.message_handler(commands=["start", "help"])
def send_welcome(message):
    bot.reply_to(message, "Hello world!")
```

```
@bot.message_handler(func=lambda message: True)
def echo_all(message):
    bot.reply_to(message, message.text)

if __name__ == "__main__":
    bot.infinity_polling()
```

В ответ на команды /start и /help бот отправит приветственное сообщение. А на все остальные сообщения от пользователя он ответит по принципу эха, то есть повторит написанное. Именно поэтому такой бот часто называют эхоботом.

`func=lambda message: True`

Код выше означает, что любое сообщение будет обработано с помощью обработчика сообщений echo_all. Его важно объявить после всех других обработчиков.

Функцию func можно сделать более сложной, чтобы она, например, проверяла наличие определённого слова или изображения в сообщении.

Для отправки сообщения пользователю используется команда send_message.

`bot.send_message(message.chat.id, 'Новое сообщение!')`

Скопируйте этот код и попробуйте повзаимодействовать с ботом, чтобы лучше понять, как он работает.

Архитектура телеграм-бота на Python

Эхоботы интересны, но они не отражают достаточно навыков, которые можно продемонстрировать в портфолио. Давайте разберёмся, как создать бота с более серьёзным функционалом.

Архитектура проекта для бота — это важная составляющая его успеха. Если она хорошо продумана, бот становится более простым в разработке и эффективным в использовании. Также благодаря этому его удобно поддерживать.

Часто при построении архитектуры используют модульную функциональность. Она позволяет создавать ботов или другие приложения с чёткой структурой и тем самым упрощает их разработку. Каждая директория такого бота выполняет определённую функцию.

Директория	**За что отвечает**
database/	Работа с базой данных. Содержит модули для подключения, выполнения запросов и работы с моделями данных
api/	Работа со сторонним сервисом через программный интерфейс (API). Содержит модули для запроса к сторонним сервисам и обработку ответов
handlers/	Обработчики сообщений. Каждый обработчик соответствует определённой команде или типу сообщения
keyboards/	Создание кнопок. Модули для генерации клавиатур с кнопками для удобного взаимодействия пользователя с ботом
states/	Состояния. Хранение и управление состояниями диалога с пользователем
utils/	Вспомогательные функции. Общие функции, используемые в разных модулях проекта
.env	Переменные окружения. Хранение конфиденциальных данных, таких как API-ключи
config.py	Конфигурационный файл. Настройка основных параметров бота, таких как токен доступа к Telegram и API
loader.py	Инициализация бота. Подключение всех необходимых модулей и запуск бота
main.py	Запуск бота из loader

Преимущества модульной функциональности:

- позволяет легко добавлять новые функции, таким образом расширяя возможности бота;
- так как каждый модуль отвечает за свою задачу, код легко читается и поддерживается.

Если вы впервые приступаете к разработке собственного бота, рекомендуем ознакомиться перед этим с репозиторием, в котором хранится пример архитектуры бота. Вы можете использовать его в качестве основы для будущего проекта.
Кроме того, чтобы лучше понять, как он устроен, советуем посмотреть видеообзор репозитория, в котором описаны все файлы и папки.

Диалог бота с пользователем

Пользователь и бот могут вести развёрнутый диалог, состоящий из множества вопросов и ответов. Чтобы не потерять важную информацию (ответы), их необходимо где-то сохранять. Самый удобный и простой метод организации диалога — использование состояний. Этот подход основан на принципе автомата состояний: мы знаем, в каком состоянии сейчас находимся, но не всегда понимаем, как мы туда попали.

Посмотрите видеопример, в котором показано составление диалога на основе состояний.

Перед тем как продолжить разбираться в этой теме, рекомендуем внимательно изучить материалы репозитория и посмотреть все видео. После этого понять следующий материал будет легче.

Работа с кнопками

Чтобы пользователю было удобнее проходить созданный вами сценарий, в чате можно использовать кнопки. В Telegram есть два основных типа кнопок: обычные и инлайн.

- **Обычные кнопки** представлены в виде JSON-объекта ReplyKeyboardMarkup. Эти кнопки используются для отправки заранее заданного текста в чат. Когда пользователь нажимает на обычную кнопку, в чат отправляется текст, который с ней связан.
- **Инлайн-кнопки** представлены в виде JSON-объекта InlineKeyboardMarkup. Они предоставляют пользователю возможность получить нужную информацию при нажатии на кнопку обратного вызова (callback). При использовании инлайн-кнопок вместо отправки текста в чат пользователь может получить заранее подготовленное сообщение, внешнюю ссылку или совершить определённое действие (например, отправить боту данные для обработки). Это делает взаимодействие бота с

пользователем более гибким, так как кнопки могут предлагать дополнительные функции и действия.

Таким образом, обычные кнопки используются для отправки текста в чат, а инлайн-кнопки позволяют пользователю выполнить определённые действия или получить дополнительную информацию, не покидая чата. Разницу между ними можно увидеть ниже.

Важное различие между обычной и инлайн-клавиатурой состоит в следующем:

- Обычная (reply) клавиатура — это заранее заданный текст, который отправляется боту. Нажатие на этот вид кнопок можно обработать с помощью декоратора message_handler.
- Инлайн-клавиатура — это отдельный объект, который добавляется под сообщением бота. Нажатие на эти кнопки можно обработать с помощью декоратора callback_query_handler.

Работа с **reply**-клавиатурой

Для работы с обычными (reply) кнопками из модуля telebot.types используются следующие классы:

- **ReplyKeyboardMarkup** — для создания объекта клавиатуры (документация).
- **KeyboardButton** — для создания кнопок клавиатуры (документация).
- **ReplyKeyboardRemove** — для удаления клавиатуры (документация).

Атрибуты, которые есть у типов Telegram Bot API, также есть у соответствующих классов в библиотеке Telebot. Кроме того, для удобства в классы добавлены дополнительные методы.

Если рассмотреть описание параметра keyboard у класса ReplyKeyboardMarkup, становится понятно, что клавиатура — это массив массивов из объектов KeyboardButton. Каждый такой массив внутри основного массива представляет собой ряд кнопок. В Python массивы представлены в виде списков (list).

Таким образом, создание клавиатуры основывается на простом алгоритме:

1. создание объектов кнопок клавиатуры;
2. создание объекта клавиатуры;
3. добавление массивов (list) нужной конфигурации с кнопками в основной массив (list) клавиатуры с помощью метода add.

Например, для построения reply-клавиатуры можно использовать следующий код:

```
from telebot.types import ReplyKeyboardMarkup, KeyboardButton,
ReplyKeyboardRemove
from telebot import TeleBot

def gen_markup():
    # Создаём объекты кнопок.
    button_1 = KeyboardButton(text="Собаки 🐕")
    button_2 = KeyboardButton(text="Кошки 🐈")

    # Создаём объект клавиатуры, добавляя в него кнопки.
    keyboard = ReplyKeyboardMarkup()
    keyboard.add(button_1, button_2)
    return keyboard

bot = TeleBot(
    ""
)  # Токен, полученный от BotFather.

@bot.message_handler(commands=["start"])
def start_message(message):
    bot.send_message(
        message.from_user.id,
        "Какое животное тебе нравится больше?",
        reply_markup=gen_markup(),  # Отправляем клавиатуру.
    )

@bot.message_handler(func=lambda message: message.text == "Собаки 🐕")
def dog_answer(message):
    bot.send_message(
        message.from_user.id,
         "Я тоже люблю собак, они так мило машут хвостиком!",
        reply_markup=ReplyKeyboardRemove(),  # Удаляем клавиатуру.
    )

@bot.message_handler(func=lambda message: message.text == "Кошки 🐈")
def cat_answer(message):
    bot.send_message(
        message.from_user.id,
        "Я тоже люблю кошек, они так умилительно мурлыкают!",
        reply_markup=ReplyKeyboardRemove(),  # Удаляем клавиатуру.
    )

bot.infinity_polling()
```

Чтобы кнопки не занимали много места, в объекте клавиатуры ReplyKeyboardMarkup можно использовать аргумент resize_keyboard=True.

```
keyboard = ReplyKeyboardMarkup(resize_keyboard=True)
```

Чтобы увидеть изменения, нужно перезапустить бота.

При нажатии на любую из кнопок клавиатура навсегда исчезнет из чата, так как в обработчиках (хендлерах) используется ReplyKeyboardRemove.

Работа с инлайн-клавиатурой

Для работы с инлайн-клавиатурой или callback-кнопками из модуля telebot.types используются следующие классы:

- **InlineKeyboardMarkup** — для создания объекта клавиатуры (документация).
- **InlineKeyboardButton** — для создания кнопок клавиатуры (документация).

Для удаления клавиатуры нет специального класса, для этого можно использовать несколько вариантов:

- Удалить сообщение.

```
bot.delete_message(callback_query.message.chat.id,
                   callback_query.message.message_id
) # Удаление сообщения.
```

- Удалить клавиатуру после нажатия.

```
bot.edit_message_reply_markup(callback_query.message.chat.id,
                              callback_query.message.message_id
                              ) # Удаление одной клавиатуры.
```

В качестве примера для построения инлайн-клавиатуры можно использовать следующий код:

```
from telebot.types import InlineKeyboardMarkup, InlineKeyboardButton
from telebot import TeleBot

def gen_markup():
    # Создаём объекты кнопок.
    button_1 = InlineKeyboardButton(text="Собаки 🐕", callback_data="dog")
    button_2 = InlineKeyboardButton(text="Кошки 🐈", callback_data="cat")


    # Создаём объект клавиатуры, добавляя в него кнопки.
    keyboard = InlineKeyboardMarkup()
    keyboard.add(button_1, button_2)
    return keyboard
```

```
bot = TeleBot(
    ""
)  # Токен, полученный от BotFather.

@bot.message_handler(commands=["start"])
def start_message(message):
    bot.send_message(
        message.from_user.id,
        "Какое животное тебе нравится больше?",
        reply_markup=gen_markup(),  # Отправляем клавиатуру.
    )

@bot.callback_query_handler(
    func=lambda callback_query: (
        callback_query.data  # Обращаемся к callback_data, указанной при создании
кнопки.
        == "dog"
    )
)
def dog_answer(callback_query):
    # Удаляем клавиатуру.
    bot.edit_message_reply_markup(
        callback_query.from_user.id, callback_query.message.message_id
    )
    # Отправляем сообщение пользователю.
    bot.send_message(
        callback_query.from_user.id,
        "Я тоже люблю собак, они так мило машут хвостиком!",
    )

@bot.callback_query_handler(
    func=lambda callback_query: (
        callback_query.data  # Обращаемся к callback_data, указанной при создании
кнопки.
        == "cat"
    )
)
def cat_answer(callback_query):
    # Удаляем клавиатуру.
    bot.edit_message_reply_markup(
        callback_query.from_user.id, callback_query.message.message_id
    )
    # Отправляем сообщение пользователю.
    bot.send_message(
        callback_query.from_user.id,
        "Я тоже люблю кошек, они так умилительно мурлыкают!",
    )
```

```
bot.infinity_polling()
```

Посмотрите, что получилось в результате работы этого кода.

Давайте потренируемся использовать новые инструменты. Для этого мы опишем сценарий для бота, по которому он сможет создавать кнопки для пользователя.

Предположим, что в сценарии будут следующие пункты:

- ввод команды /create_markup, где бот предлагает сформировать инлайн-клавиатуру;
- ввод количества кнопок;
- ввод текста и callback информации для каждой кнопки;
- информирование о созданной клавиатуре и её отправка;
- обработка нажатий на клавиатуру.

```
from telebot.types import InlineKeyboardMarkup, InlineKeyboardButton
from telebot import TeleBot
from telebot.handler_backends import State, StatesGroup
from telebot.custom_filters import StateFilter

# Состояния для диалога.
class KeyboardsState(StatesGroup):
    buttons_count = State()
    keyboard_text_and_callback = State()
    send_keyboard = State()

bot = TeleBot(
    ""
)  # Токен, полученный от BotFather.

def gen_markup(buttons_info):
    # Создаём объект клавиатуры.
    keyboard = InlineKeyboardMarkup()


    for text_keyboard, callback_keyboard in buttons_info.items():
        # Создаём объект кнопки и добавляем её к клавиатуре.
        button = InlineKeyboardButton(
            text=text_keyboard, callback_data=callback_keyboard
        )
        keyboard.add(button)

    return keyboard

@bot.message_handler(commands=["create_markup"])
def handle_start_message(message):
```

```
    # Присваиваем состояние.
    bot.set_state(message.from_user.id, KeyboardsState.buttons_count,
message.chat.id)
    bot.send_message(
        message.from_user.id,
        "Привет! Я помогу тебе сформировать инлайн-клавиатуру\nВведи
количество кнопок для будущей клавиатуры",
    )

# Ловим его в одном или нескольких хендлерах.
@bot.message_handler(state=KeyboardsState.buttons_count)
def handle_buttons_count(message):
    if message.text.isdigit():
        bot.set_state(
            message.from_user.id,
            KeyboardsState.keyboard_text_and_callback,
            message.chat.id,
        )


        with bot.retrieve_data(message.from_user.id) as data:
            # Сохраняем информацию и делаем заготовки объектов, которые
понадобятся для нашего сценария.
            data["buttons_count"] = int(message.text)
            data["buttons"] = {}
            data["temp_keyboard_data"] = []
        bot.send_message(
            message.from_user.id, "Отлично! Теперь введи текст для кнопки"
        )
    else:
        bot.send_message(message.from_user.id, "Количество кнопок должно быть
указано числом!")

@bot.message_handler(state=KeyboardsState.keyboard_text_and_callback)
def handle_keyboard_text_and_callback(message):
    # Удерживаем пользователя в этом состоянии, пока не получим всё, что
нужно.
    with bot.retrieve_data(message.from_user.id) as data:
        temp_keyboard_data = data["temp_keyboard_data"]


        if not temp_keyboard_data:
            temp_keyboard_data.append(message.text)
            bot.send_message(
                message.from_user.id, f"Укажи callback для кнопки {message.text}"
            )
        elif len(temp_keyboard_data) == 1:
            temp_keyboard_data.append(message.text)
```

```
            key, value = temp_keyboard_data
            data["buttons"][key] = value
            temp_keyboard_data.clear()
            data["buttons_count"] -= 1


            if data["buttons_count"]:
                bot.send_message(
                    message.from_user.id,
                    "Данные сохранены! Введи  текст для следующей кнопки",
                )
            else:
                bot.set_state(message.from_user.id, KeyboardsState.send_keyboard)
                bot.send_message(
                    message.from_user.id,
                    "Кнопки, которые ты хотел создать",
                    reply_markup=gen_markup(data["buttons"]),
                )

# В callback_query_handler ловим состояние. Так как нам не нужен
дополнительный фильтр, указываем функцию как None.
@bot.callback_query_handler(func=None, state=KeyboardsState.send_keyboard)
def handle_buttons(callback_query):
    with bot.retrieve_data(callback_query.from_user.id) as data:
        pressed_button_name = None
        for key, value in data["buttons"].items():
            if callback_query.data == value:
                bot.answer_callback_query(
                    callback_query.id,
                    f"Поймал нажатие на кнопку {key}",
                    show_alert=True,
                )
                pressed_button_name = key


        data["buttons"].pop(pressed_button_name, None)
        bot.edit_message_reply_markup(
            callback_query.from_user.id,
            callback_query.message.message_id,
            reply_markup=gen_markup(data["buttons"]),
        )
    if not data["buttons"]:
        bot.send_message(callback_query.from_user.id, "Все кнопки были нажаты!")
        bot.delete_state(callback_query.from_user.id)


# Перед запуском бота подключаем фильтр состояний.
```

```
bot.add_custom_filter(StateFilter(bot))
bot.infinity_polling()
```

В этом материале вы узнали, как создать и настроить простого телеграм-бота с помощью библиотеки pyTelegramBotAPI. Сейчас ваш бот может взаимодействовать с пользователями с помощью текстовых сообщений и кнопок. В следующих темах вы узнаете, как добавить в него новый функционал. Это позволит сделать бот ещё более полезным и интересным для пользователей.