Запуск бота

В этом руководстве мы подробно расскажем, как развернуть Telegram-бота, написанного на Python, с использованием Docker. Мы пройдем через все шаги, начиная с клонирования репозитория на GitHub и заканчивая запуском. Давайте начнем.

Шаг 1: Клонирование репозитория

Первый шаг — это клонирование репозитория с кодом нашего Telegram-бота с GitHub. Для этого откройте терминал и выполните следующую команду:

git clone https://github.com/USERNAME/REPO_NAME.git

Замените `USERNAME` на имя пользователя и `REPO_NAME` на название репозитория.

Пример: **git clone https://github.com/ichwillnicht/kommtanzen.git**

После завершения клонирования перейдите в папку проекта:

cd botvacation

Шаг 2: Активация виртуального окружения

Теперь мы активируем виртуальное окружение, что поможет управлять зависимостями проекта. Для этого выполните следующие команды:

На Windows:

python -m venv venv
venv\Scripts\activate

На macOS и Linux:

python3 -m venv venv
source venv/bin/activate

После активации виртуального окружения вы увидите, что название окружения появилось в начале строки вашего терминала.

Шаг 3. Установка зависимостей

В нашем проекте бота используются сторонние библиотеки. Чтобы другим разработчикам было проще установить их у себя, зависимые пакеты обычно указываются в файле requirements.txt

Чтобы установить все библиотеки нужных версий, необходимо в активированном виртуальном окружении (перед путями указано (venv)) - выполнить команду:

pip install -r requirements.txt

Шаг 3: Установка Docker

Если у вас еще не установлен Docker, скачайте и установите его с официального сайта: Docker. Следуйте инструкциям на сайте для вашей операционной системы.

Для Windows качаем AMD64 (ARM - версия для мобильных устройств).

После установки проверьте, что Docker работает, выполнив команду:

docker --version

Шаг 4: Запуск контейнеров с помощью Docker Compose

В корне проекта есть файл `docker-compose.yml`. Этот файл содержит конфигурацию для запуска контейнеров. Пока что мы не будем погружаться в процесс работы контейнеров подробно, просто упомянем, что контейнеры необходимы, чтобы мы могли проводить операции с базой данных. Такой опыт максимально приблизит нас к работе с реальными проектами, запущенными на сервере.

Чтобы запустить контейнеры, в терминале выполните следующую команду:

docker-compose up -d

Параметр `-d` запускает контейнеры в фоновом режиме - и мы можем писать в этом же терминале команды непосредственно для бота.

Шаг 5: Настройка API_TOKEN

Следующий шаг — это настройка API_TOKEN. Создайте файл `.env`
`Он должен находиться в той же папке, что и файл main.py`
Внутри файла создайте переменную `API_TOKEN = “__”`
Замените значение на ваш собственный токен, который вы получили от BotFather в Telegram.

Пример строки в файле `.env`:

API_TOKEN=”`588:AAE`”

Шаг 6: Запуск бота

Теперь, когда предварительные настройки завершены, вы можете запустить вашего Telegram-бота. В терминале выполните команду:

python main.py

Если все сделано правильно, вы увидите курсор на следующей строке - бот ожидает запуска в Telegram.

РАБОТА С БАЗОЙ - файл dbcreate

В этом руководстве мы рассмотрим, как использовать файл `dbcreate` для работы с базой данных PostgreSQL в вашем боте. Мы будем использовать ORM библиотеку SQLAlchemy, которая позволяет легко взаимодействовать с базой данных, не углубляясь в написание SQL-запросов.

Зачем использовать ORM?

ORM (Object-Relational Mapping) позволяет работать с базой данных, используя привычные классы и функции на Python. Вместо написания сложных SQL-запросов, вы можете создавать модели, которые представляют ваши таблицы, и использовать методы для добавления, редактирования и удаления данных.

Основные возможности

1. Создание новых моделей

Если вы хотите добавить новую функциональность в вашего бота, вы можете создать новую модель. Например, если вы хотите добавить возможность хранения событий, вы можете создать класс `Event`:

```
class Event(Base):
    __tablename__ = 'events'
    id = Column(Integer, primary_key=True, index=True)
    title = Column(String, nullable=False)
    description = Column(Text, nullable=True)
    date = Column(DateTime, nullable=False)
    user_id = Column(Integer, ForeignKey('users.id'), nullable=False)
    user = relationship('User', back_populates='events')
```

Не забудьте добавить обратную связь в класс `User`:

events = relationship('Event', back_populates='user')

2. Добавление новых функций

После создания модели, вы можете добавить функции для работы с ней, например, для добавления, редактирования и удаления событий, аналогично тому, как это сделано для заметок.

Для получения всех событий пользователя вы можете создать функцию, аналогичную функции `get_user_notes`:

```
async def get_user_events(user_chat_id: int):
    async with async_session() as session:
        result = await session.execute(
            select(Event).where(Event.user_id == select(User.id).where(User.chat_id == 
user_chat_id))
        )
        events = result.scalars().all()
        return events
```

3. Добавление нового в json

json файлы отвечают за список ваших вещей при сборе. В любой момент вы можете редактировать его или добавлять что-то новое.
Обратите внимание: ВАЖНО делать все по инструкции, чтобы не нарушить работу всего бота.
Вот один из списков

```
[
    "1.",
    "Загранпаспорт, паспорт, виза"
],
[
    "2.",
    "Медицинская страховка"
],
[
    "3.",
    "Билеты"
],
[
    "4.",
    "Права"
],
[
    "5.",
    "Бронь отеля"
],
```

```
 [
    "6.",
    "Наличные"
 ]
]
```

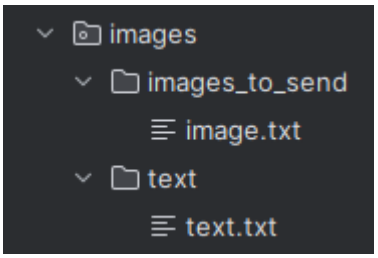
Вы можете дублировать блоки, которые отвечают за одну вещь.
Самый последний блок не должен иметь запятую в конце.

```
[
    "6.",
    "Наличные"
]
```

Добавление новых файлов json требует более сложной настройки.

Это может быть добавлено позже.

4. Добавление текста и картинок при рассылке.



За картинки отвечает файл image.txt
Хранение в нем реализовано в виде ссылок на изображения. Вы можете добавлять и удалять их, по своему усмотрению. Ограничений никаких нет.

Тоже самое касается файла text.txt
Все, что нужно учесть: Каждая фраза и url должны начинаться с новой строки.