Основы разработки надёжного бота

Виртуальное окружение в Python

Для создания качественного, надёжного и функционального бота вам потребуется:

1. создать виртуальное окружение;
2. обеспечить безопасность данных пользователей;
3. интегрировать API в телеграм-бота;
4. настроить работу с базами данных;
5. применить технологии ORM, связывающие базу данных с объектами языка Python.

В этом материале вы рассмотрите первые три пункта списка, а о работе с базами данных и ORM узнаете в отдельных материалах.

Виртуальное окружение в Python — изолированная среда, которая содержит собственную копию интерпретатора Python, библиотек и других зависимостей.

Лучше понять этот термин поможет аналогия. Представьте, что у вас есть отдельный кабинет для работы или учёбы. В нём есть всё необходимое для выполнения текущих задач и нет отвлекающих факторов: ни шумных соседей, ни посторонних вещей, ни аппетитных запахов. Точно так же виртуальное окружение в Python создаёт свою собственную «комнату», где все ресурсы и инструменты находятся под контролем, без возможных конфликтов с другими программами или библиотеками. Это помогает обеспечить чистоту и надёжность работы вашего кода, изолируя его от внешних факторов.

Зачем использовать виртуальное окружение

- **Изоляция.** Виртуальное окружение позволяет изолировать зависимости проекта от глобального окружения Python. Это означает, что вы можете использовать разные версии библиотек для разных проектов, не опасаясь конфликтов.
- **Повторяемость.** Виртуальное окружение гарантирует, что ваш код будет работать одинаково на разных машинах, поскольку все зависимости будут установлены в одинаковой среде.
- **Управление зависимостями.** Виртуальное окружение позволяет легко управлять зависимостями вашего проекта. Вы можете добавлять, удалять и обновлять библиотеки, не влияя на другие проекты.
- **Порядок.** Виртуальное окружение помогает поддерживать порядок в проектах. Все зависимости проекта будут находиться в одном месте, что упрощает их поиск и управление.

Как создать виртуальное окружение

Для создания виртуального окружения потребуется использовать модуль venv, который встроен в Python.

1. Откройте командную строку.
2. Перейдите в каталог с проектом, где вы хотите создать виртуальное окружение.
3. Выполните следующую команду:

```
python -m venv venv
```

1. Эта команда создаст папку venv с виртуальным окружением.

Имя папки с виртуальным окружением может быть любым. Негласный стандарт для имён окружений — **env** и **venv.**

Как активировать/деактивировать виртуальное окружение

Чтобы активировать виртуальное окружение, необходимо выполнить команду активации.

Для Windows:

```
venv\Scripts\activate.bat
```

Для Linux/macOS:

```
source venv/bin/activate
```

Чтобы деактивировать виртуальное окружение, выполните команду:

```
deactivate
```

Теперь ваше виртуальное окружение готово к дальнейшему использованию.

Хранение секретных данных

Секретные данные — это информация, которая всегда должна оставаться конфиденциальной. Она может включать в себя пароли, ключи API, токены доступа и прочие личные данные, которые используются для аутентификации или авторизации в системе.

Разберём некоторые термины, упомянутые выше.

Аутентификация — проверка личности пользователя. Когда вы вводите свой логин и пароль, система проверяет их, чтобы удостовериться, что это действительно вы.

Авторизация — определение прав доступа после успешной аутентификации. Например, после ввода логина и пароля в банковском приложении система определяет, можете ли вы видеть баланс, переводить деньги или выполнять другие действия.

Хранение секретных данных в коде напрямую — плохая практика по двум причинам:

1. **Безопасность.** Если секретные данные хранятся в исходном коде, их легко могут обнаружить злоумышленники при просмотре вашего кода. Это создаёт уязвимость для проекта.
2. **Контроль доступа.** Если разработчики имеют доступ к исходному коду, они могут получить доступ к секретным данным. Это создаёт риск для безопасности данных.

Для безопасного хранения секретных данных в Python-проекте можно использовать библиотеку python-dotenv. Она позволяет загружать переменные окружения из файла .env в корне вашего проекта.

Переменные окружения — значения, доступные в рамках конкретной среды выполнения программы или приложения. Они обычно используются для хранения конфиденциальной информации или конфигурационных параметров и не должны быть явно указаны в исходном коде программы.

Пример использования библиотеки python-dotenv:

1. Установите библиотеку с помощью pip:

```
pip install python-dotenv
```

Создайте файл .env в корне своего проекта и добавьте в него секретные данные в формате KEY=VALUE.

Например:

```
API_KEY='abqwecq'
BOT_TOKEN='zxcqwe9sdq'
```

В своём конфигурационном файле config.py загрузите переменные окружения из файла .env с помощью load_dotenv:

```
from dotenv import load_dotenv
import os
load_dotenv()
API_KEY = os.getenv("API_KEY")
BOT_TOKEN = os.getenv("BOT_TOKEN")
```

Теперь ваши секретные данные хранятся в файле .env, который необходимо добавить в .gitignore, чтобы избежать его попадания в репозиторий. Таким образом, вы обеспечиваете безопасное и удобное управление секретными данными в своём Python-проекте.

Так как файл .env в репозиторий не добавляют, для переменных окружения добавляют шаблон для заполнения .env.template с переменными и кратким описанием, которые нужны для работы. Тем самым вы даёте понять тем, кто

использует ваш проект, что нужно создать .env по образу и подобию .env.template.

Пример содержимого .env.template:

```
API_KEY=здесь_ваш_ключ_доступа_к_api
BOT_TOKEN=здесь_ваш_ключ_доступа_к_боту
```

Внедрение переменных окружения обеспечивает безопасное хранение конфиденциальных данных в проекте Python, предотвращая их случайное раскрытие и повышая уровень безопасности приложения. Этот подход не только защищает данные, но и обеспечивает гибкость в управлении конфигурацией проекта.

Работа с **API**

API (англ. Application Programming Interface — программный интерфейс приложения) — набор способов и правил, по которым разные программы общаются между собой и обмениваются данными.

Вы сталкиваетесь с API, когда оплачиваете покупку банковской картой. После того как вы прикладываете карту, терминал отправляет запрос к API банка: «Клиент хочет купить билет». Банк принимает этот запрос, проверяет количество денег на счёте и отвечает: «Всё в порядке. Пусть покупает». В рамках финальной работы курса в роли терминала выступает телеграм-бот, а в роли банка — выбранный вами сервис, например Яндекс Погода.

Какие-то API полностью бесплатные, в каких-то нужно получить специальный ключ. Чаще встречается второй вариант, так как нагрузка на бесплатные API очень высокая, а с помощью ключа сервис может ограничить запросы.

Яндекс предоставляет огромное количество API, которые вы можете использовать в своих проектах. Также существуют агрегаторы API от сервисов, например Public APIs, так что всегда можно найти подходящий вариант.

Выполнение пробных запросов

Потренируйтесь во взаимодействии с одним из них. Возьмите API Яндекс Словаря. Для начала нужно получить бесплатный ключ на странице сервиса.

В документации на странице API Яндекс Словаря есть два эндпоинта — ссылки, по которым можно запросить данные: getLangs и lookup. Там же можно посмотреть примеры запросов.

Перейдя по ссылке ниже, вы можете увидеть JSON со списком доступных направлений перевода: https://dictionary.yandex.net/api/v1/dicservice.json/getLangs?key=ВАШ_API_КЛЮЧ

Не забудьте добавить полученный API-ключ. Пример ссылки с указанным API-ключом: https://dictionary.yandex.net/api/v1/dicservice.json/getLangs?key=dict.1.1.20240321T080836ZPPPZ.722893209af57535.4d78957b4b15afa302ca7dd0785d004a67essqca414

Посмотрим, что с этим можно сделать в методе lookup. Здесь требуется указать дополнительные параметры: направление перевода и текст. Также есть необязательные: язык интерфейса и опции поиска.

Для примера возьмите слово «задача» и направление ru-en. Вставьте их в URL-параметры: https://dictionary.yandex.net/api/v1/dicservice.json/lookup?key=ВАШ_API_КЛЮЧ&lang=ru-en&text=задача

В этой ссылке также необходимо добавить полученный ранее API-ключ. Пример ссылки с указанным API-ключом: https://dictionary.yandex.net/api/v1/dicservice.json/lookup?key=dict.1.1.20240321T080836ZPPPZ.722893209af57535.4d78957b4b15afa302ca7dd0785d004a67essqca414&lang=ru-en&text=задача

В ответ вы получите разные переводы и синонимы. Проделайте такие же действия у себя в браузере.

Выполнение запросов с помощью **requests**

Теперь попробуйте сделать то же самое с помощью Python. Для начала установите модуль requests — он понадобится для отправки запросов:

```
pip install requests
```

Напишите следующий код:

```
import requests
from pprint import pprint

API_KEY = 'ВАШ_API_КЛЮЧ'
BASE_URL = 'https://dictionary.yandex.net/api/v1/dicservice.json'

def get_langs():
    response = requests.get(f'{BASE_URL}/getLangs', params={
        'key': API_KEY
    })
    return response

def lookup(lang, text, ui='ru'):
    response = requests.get(f'{BASE_URL}/lookup', params={
        'key': API_KEY,
        'lang': lang,
```

```
        'text': text,
        'ui': ui
    })
    return response

langs_response = get_langs()
if langs_response.status_code != 200:
    print('Не удалось получить список направлений перевода')
    exit(1)

langs = langs_response.json()
print('Выберите одно из доступных направлений перевода')
print(langs)
while (lang := input('Введите направление: ')) not in langs:
    print('Такого направления нет. Попробуйте ещё раз')

text = input('Введите слово или фразу для перевода: ')
lookup_response = lookup(lang, text)
if lookup_response.status_code != 200:
    print('Не удалось выполнить перевод:', lookup_response.text)
    exit(1)

pprint(lookup_response.json())
```

На что стоит обратить внимание:

- BASE_URL — часть ссылки, которая не меняется при обращениях к этому API.
- Запрос делается с помощью requests.get, где первый аргумент — это ссылка, а дальше вы указываете URL-параметры, передав их в params.

В сырой ссылке URL-параметры выглядят так:

```
?key=ВАШ_API_КЛЮЧ&lang=ru-en&text=задача
```

В принципе, если вставить их в код таким образом, он всё равно сработает.

В виде словаря код выглядит так:

```
params = {
    'key': 'ВАШ_API_КЛЮЧ',
    'lang': 'ru-en',
    'text': 'задача'
}
```

Использование словаря повысит читаемость кода, поэтому этот способ предпочтителен.

- Ещё есть requests.post, который посылает POST-запрос.
  Если GET-запрос предназначен для получения информации от сервера, то POST —

больше для передачи информации серверу. Применяйте тот или иной метод в зависимости от документации.
- Методы requests.get и requests.post возвращают объект ответа, который содержит в себе код (2xx — успешные запросы, 4xx — неправильные запросы, 5xx — ошибки сервера), а также текст. Если обращаетесь в формате JSON, вы можете сразу получить его с помощью response.json().

В API некоторые данные могут передаваться в заголовках или теле запроса:

```
requests.post(url, headers={}, data={})
Часто в API указывают ключ в заголовке:
requests.post(url, headers={
    'X-RapidAPI-Key': 'ВАШ_API_КЛЮЧ',
    'X-RapidAPI-Host': 'hotels4.p.rapidapi.com'
}, data={
    'currency': 'USD'
})
```

У всех API разный контракт взаимодействия, поэтому тщательно изучайте документацию.

Интеграция API

Перейдём к заключительной части этого материала — связке телеграм-бота и API словаря с помощью полученных знаний.

Посмотрите на пример готового решения.

На что обратить внимание

Так как бот относительно небольшой, весь код разбит на четыре модуля:

- config.py — здесь объявляются токены, ключи и другие константы;
- states.py — здесь содержатся состояния пользователя;
- api.py — отвечает за взаимодействие с API;
- main.py — здесь создаются обработчики и запускается бот.

В идеале в main.py должен оставаться только запуск бота, а обработчики имеет смысл выносить в отдельный модуль.

Конфигурационный файл

Replit поддерживает хранение переменных окружения через меню Secrets, что позволяет получать эти значения с помощью os.getenv.

В PyCharm для этого нужно создать файл .env и загрузить его в переменные программы с помощью библиотеки python-dotenv.

Состояния

У бота всего три состояния:

- base — пользователь находится в главном меню;
- lang — пользователь выбирает направление перевода;
- lookup — пользователь ищет слово или фразу.

Работа с API

Здесь применён простой интерфейс: функции делают запрос и возвращают обычные словари или списки, которые получены от сервера.

Обычно сервер возвращает лишнюю информацию, поэтому её лучше фильтровать в модуле api.py. Например, это можно сделать, создав класс вида LookupResult, который в более удобном виде будет хранить списки синонимов и переводов.

Основная логика телеграм-бота

1. Так как доступные языки не меняются, их можно запросить сразу при запуске. Это повысит производительность, так как не нужно будет ожидать ответа от сервера.
2. При вводе команды /start пользователь переходит в базовое состояние base.
3. При вводе команды /set_lang пользователь переходит в состояние lang. В функции-обработчике состояния lang бот сохраняет то, что ввёл пользователь, и возвращает его (пользователя) в состояние base.
4. При вводе команды /lookup пользователь переходит в состояние lookup. После поиска пользователь снова возвращается в состояние base.
5. При выполнении запросов к API мы распечатывали словарь, который вернул сервер. Пользователю будет неудобен такой формат, так как не все пользователи — программисты. Поэтому лучше вывести словарь в более читаемом виде, обработав полученные данные.
   Для этого имеет смысл воспользоваться модулем json и функцией json.dumps, которая позволит:
   - разбить полученные данные с помощью отступов;
   - представить ответ в виде HTML-разметки.
     Чтобы в боте корректно отображалась HTML-разметка, следует использовать тег <pre>.
6. С помощью set_my_commands задаются основные команды, к которым пользователь может обратиться. Так это выглядит в Telegram:

Вы познакомились с созданием и настройкой телеграм-бота, интегрированного с внешним API словаря. Полученные знания позволят вам создавать более сложных и функциональных ботов, обеспечивая удобство их использования.