УДК 004.9

Разработка REST API на Flask

Пастушенко Владислав Александрович
Брянский государственный университет имени академика И.Г. Петровского
магистрант

Аннотация
В статье рассмотрены понятия REST и API, методы и принципы разработки RESTful API. Рассмотрен пример реализации API на языке программирования Python и фреймворке Flask.
Ключевые слова: Python, Flask, REST, API, разработка, серверные приложения.

Development of REST API on Flask

Pastushenko Vladislav Aleksandrovich
Bryansk state university named by academician I.G. Petrovsky
master student

Abstract
This article describes the concepts of REST and API, methods and principles of RESTful API development. An example of API implementation in Python programming language and Flask framework is considered.
Keywords: Python, Flask, REST, API, development, server applications.

API – это интерфейс прикладного программирования и относится к видам связи между любыми двумя приложениями. API – это просто средство, которое позволяет двум программным сущностям общаться друг с другом.

Такие компании, как Google, Яндекс и многие другие, имеют API, которые позволяют внешним приложениям использовать свои функциональные возможности, не раскрывая свою кодовую. Существует высокая вероятность того, что компания, с которой вы хотите работать, уже имеет API-интерфейс – как для разработчиков, так и для конечных пользователей.

Но почему компании позволяют нам использовать их контент через API? Предоставляя пользователям доступ к своему контенту, компании повышают ценность как для разработчиков, так и для пользователей. Вместо создания новой функциональности с нуля и повторного изобретения колеса разработчики могут использовать существующие API и сосредоточиться на своих основных целях. Эта практика на самом деле помогает компаниям путем построения отношений с разработчиками и расширения их пользовательской базы.

Как и API, REST – это аббревиатура, и она означает передачу репрезентативного состояния. Это архитектурный стиль для разработки стандартов общения между компьютерами, что упрощает взаимодействие систем друг с другом. Проще говоря, REST – это набор правил, которым разработчики следуют при создании API. Система называется RESTful, когда она придерживается этих ограничений.

Чтобы лучше понять, что такое RESTful API, нам нужно определить, что означают термины “клиент” и “ресурс”.

- Клиент: клиент может обратиться к приложению, которое использует API. При использовании API Яндекс Карт в приложении вы получаете доступ к ресурсам через API, что делает вас клиентом. Кроме того, веб-браузер также может быть клиентом.
- Ресурс: ресурс описывает объект, данные или часть информации, которые могут потребоваться для хранения или отправки в другие службы. Например, координаты местоположения, которые вы получаете при работе с API Яндекс Карт, являются ресурсом.

Поэтому, когда клиент отправляет запрос на сервер, он получает доступ к ресурсу. Но какой язык используют клиенты и серверы?

Чтобы люди могли говорить друг с другом, у нас есть правильный синтаксис и грамматика. Без них невозможно понять, что передается. Аналогично, API-интерфейсы имеют набор правил для взаимодействия компьютеров или программ друг с другом, которые называются протоколами.

HTTP является одним из протоколов, который позволяет получать данные. Это основа любой передачи данных в интернете и протокол клиент-серверного взаимодействия. RESTful API почти всегда реализуются на HTTP.

Когда мы работаем с RESTful API, клиент отправляет HTTP-запрос, а сервер отвечает HTTP-ответом. Углубимся в то, что HTTP-запросы и HTTP-ответы собой представляют.

Когда HTTP-запрос отправляется на сервер, он обычно содержит следующие блоки:

- Заголовок;
- Пустая строка, разделяющая заголовок и тело;
- Необязательное тело.

Заголовок состоит из команды HTTP, URI и номера версии HTTP, который в совокупности называется строкой запроса.

Листинг 1

```
GET /home.html HTTP/1.1
```

В приведенном выше листинге “GET” – это HTTP-команда “home.html” – это URI, из которого мы хотим получить данные, а “HTTP/1.1” относится к версии HTTP.

“GET” – это не единственный HTTP-метод, поэтому давайте посмотрим на некоторые другие часто используемые HTTP-метод.

– GET –используется только для получения информации с сервера. Запросы, использующие этот метод, должны только запрашивать данные и не должны никак их изменять.
  – POST – используется для отправки данных на сервер с помощью форм.
  – PUT – используется для обновления данных ресурса.
  – DELETE – удаляет все текущие представления целевого ресурса.

Когда сервер получает запрос, он отправляет сообщение обратно клиенту. Если запросы успешны, он возвращает запрошенные данные, иначе он вернет ошибку.

Когда HTTP-ответ отправляется обратно клиенту, он обычно содержит следующее:

– Заголовок;
– Пустая строка, отделяющая заголовок от тела;
– Необязательное тело.

В данном случае заголовок содержит версию HTTP, код состояния и сообщение, которое объясняет код состояния на обычном языке.

Существует много кодов, передаваемых между сервером и клиентом. Некоторые из наиболее распространенных:

– 200 OK – запрос выполнился успешно;
– 201 Created – в результате запроса успешно создан ресурс;
– 400 Bad Request – запрос не может быть обработан из-за неверного синтаксиса запроса;
– 404 Not Found – сервер не смог найти запрошенный ресурс или страницу.

Рассмотрим реализацию RESTful API на языке Python и фреймворке Flask. В качестве примера возьмем API личной библиотеки. Например, мы имеем сущность Book (листинг 2) с полями: id, title, author, genre.

Листинг 2

```
class Book(Base):
  __tablename__ = 'book'

  id = Column(Integer, primary_key=True)
  title = Column(String(250), nullable=False)
  author = Column(String(250), nullable=False)
  genre = Column(String(250))

  @property
  def serialize(self):
     return {
        'title': self.title,
        'author': self.author,
        'genre': self.genre,
```

```
        'id': self.id,
    }
```

Далее необходимо описать функции контроллеры, которые будут обрабатывать запросы на необходимые URL (листинг 3).

Листинг 3

```
@app.route("/")
@app.route("/booksApi", methods = ['GET', 'POST'])
def booksFunction():
   if request.method == 'GET':
       return get_books()
   elif request.method == 'POST':
       title = request.args.get('title', '')
       author = request.args.get('author', '')
       genre = request.args.get('genre', '')
       return makeANewBook(title, author, genre)

@app.route("/booksApi/<int:id>", methods = ['GET', 'PUT',
'DELETE'])
def bookFunctionId(id):
   if request.method == 'GET':
       return get_book(id)

   elif request.method == 'PUT':
       title = request.args.get('title', '')
       author = request.args.get('author', '')
       genre = request.args.get('genre', '')
       return updateBook(id,title, author,genre)

   elif request.method == 'DELETE':
       return deleteABook(id)
```

Мы создали две функции – booksFunction() и bookFunction(id). Первая обрабатывает запросы на URL “/” или “/booksApi” с HTTP-методами GET и POST. Когда функция получает GET запрос, она возвращает результат выполнения функции get_books(), которую мы опишем позже. При POST запросе контроллер вернет результат выполнения функции makeANewBook с параметрами из HTTP-запроса.

Вторая функция контроллер, также обрабатывает GET запрос, но на URL “/booksApi/<int:id>” и возвращает результат вызова функции get_books(id) но с параметром “id”.

Также функция контроллер bookFunctionId(id) обрабатывает PUT и DELETE HTTP-методы, которые вызывают функции updateBook() и deleteBook() соответственно.

В листинге 4 приведены функции, которые вызываются контроллерами для получения необходимых ресурсов, а также их создания, обновления и удаления. Результатом выполнения этих функций является HTTP-ответ, с телом в формате JSON.

Листинг 4

```
from Flask import jsonify
def get_books():
   books = session.query(Book).all()
   return jsonify(books= [b.serialize for b in books])

def get_book(book_id):
   books = session.query(Book).filter_by(id = book_id).one()
   return jsonify(books= books.serialize)

def makeANewBook(title,author, genre):
   addedbook = Book(title=title, author=author,genre=genre)
   session.add(addedbook)
   session.commit()
   return jsonify(Book=addedbook.serialize)

def updateBook(id,title,author, genre):
   updatedBook = session.query(Book).filter_by(id = id).one()
   if not title:
       updatedBook.title = title
   if not author:
       updatedBook.author = author
   if not genre:
       updatedBook.genre = genre
   session.add(updatedBook)
   session.commit()
   return "Updated a Book with id %s" % id

def deleteABook(id):
   bookToDelete = session.query(Book).filter_by(id = id).one()
   session.delete(bookToDelete)
   session.commit()
   return "Removed Book with id %s" % id
```

Как можно заметить, создание RESTful API не является сложной задачей. А при использовании современных языков программирования и их фреймворков данная задача решается стандартными средствами.

Библиографический список

1. Архитектура REST URL: https://habr.com/ru/post/38730/ (Дата обращения 08.07.2019)
2. Designing a RESTful API with Python and Flask URL: https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask (Дата обращения 08.08.2019)
3. Web Security: введение в HTTP URL: https://habr.com/ru/company/edison/blog/433288/ (Дата обращения 08.08.2019)