



Информационные ресурсы в финансовом мониторинге

НИЯУ МИФИ, КАФЕДРА ФИНАНСОВОГО МОНИТОРИНГА

КУРС ЛЕКЦИЙ

В.Ю. РАДЫГИН. ЛЕКЦИЯ 3

Часть 1

REST API

REST API

REST (Representational State Transfer) – это некоторый стандарт описания архитектуры прикладных интерфейсов для распределенных систем (обычно веб-приложений). Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.

В отличие от SOAP-стандарта, данные в REST не заворачиваются в дополнительный слой (в SOAP – это XML). Роль данного слоя играют комбинация HTTP-адреса и HTTP-метода действия.

HTTP-адрес как идентификатор объекта

REST API базируются на протоколе HTTP. Причём HTTP-адрес идентифицирует ту единицу информации, с которой необходимо осуществлять работы. Говоря другим языком, сам адрес объекта и является его идентификатором.

Пример

Пусть у нас есть информация о клиентах (Client) и их счетах (Account). Тогда клиенту с ключом 21 будет соответствовать адрес:

`/clients/21`

А банковскому счёту клиента 21 с ключом 40040810300008900123 будет соответствовать адрес:

`/clients/21/accounts/40040810300008900123`

Адреса и методы REST-протокола

Действие	Метод	Путь	Содержит ли запрос доп. информацию
Получение информации о списке объектов типа Client	Get	/clients	Нет
Получение информации о одном объекте типа Client (обычно, эта информация более подробная, чем в списке объектов) со значением ключа key	Get	/clients/key	Нет
Создание нового объекта Client	Post	/clients	Да
Редактирование объекта Client с ключом key	Put	/clients/key	Да
Удаление объекта Client с ключом key	Delete	/clients/key	Нет
Все объекты Account, принадлежащие объекту Client с ключом key	Get	/clients/key/accounts	Нет
Редактирование объекта Account с ключом key_a, принадлежащего объекту Client с ключом key_c	Put	/clients/key_c/accounts/key_a	Да
...

Дополнительные методы REST API

Действие	Метод	Путь	Содержит ли запрос доп. информацию
Проверка существования метода для списка объектов Client (ответ будет содержать только заголовок, без тела)	Head	/clients	Нет
Проверка существования метода для объекта Client с ключом key (ответ будет содержать только заголовок, без тела)	Head	/clients/key	Нет
Получение списка методов, доступных по данному адресу. Перечень методов будет в заголовке Allow.	Options	/clients	Нет

Ответы на запросы к REST API

На предыдущем слайде мы рассмотрели, как клиент строит обращения к REST API серверу. Но мы никак не рассматривали ответы на данные запросы. В каком формате они будут получены?

REST-стандарт никак не специфицирует формат передачи ответа на запрос и того, в какой форме отправляются сопроводительные к запросам данные. Но он стандартизирует процесс того, как можно сообщить об этом от одной стороны обмена данными к другой.

MIME-типы заголовков

При обращении к серверу REST API клиент посредством HTTP-протокола может передать специальный заголовок `Accept`, с указанием MIME-типа ожидаемого ответа. Например:

`Accept: application/json` – заголовок, означающий, что ожидается получение ответа в формате JSON.

Любой запрос или ответ может содержать заголовок `Content-Type`, с указанием MIME-типа своего содержимого. Обычно также указывается кодировка символов. Например:

`Content-Type: application/json; charset=UTF-8`

Обычно REST API используют преимущественно JSON-формат обмена данными.

Некоторые виды MIME-типов

Группа	Тип	Содержимое
text	text/plain	Обычный текст
text	text/css	CSS-таблица стилей
text	text/html	HTML-страница
image	image/png	Картинка в формате PNG
image	image/jpeg	Картинка в формате JPEG
image	image/gif	Картинка в формате GIF
audio	audio/wav	Звуковой файл с кодировкой WAV
audio	audio/mpeg	Звуковой файл с кодировкой MPEG
video	video/mp4	Видео файл с кодировкой MPEG4
application	application/json	JSON-документ
application	application/pdf	PDF-документ
application	application/xml	XML-документ

Код ответа HTTP

HTTP-ответ может содержать код состояния HTTP (англ. HTTP status code). Значение данного кода может являться одним из способов сигнализировать об ошибке исполнения запроса или о его успешности.

- Коды ответов подразделяются на несколько групп:
- 1XX — информационные;
- 2XX — информируют о случаях успешного принятия и обработки запроса клиента;
- 3XX — сообщают клиенту, что для успешного выполнения операции необходимо сделать другой запрос, как правило по другому URI;
- 4XX — ошибка клиента;
- 5XX — ошибка сервера. Возвращается клиенту в случае неудачного выполнения операции по вине сервера.

Наиболее популярные коды

404 Not Found – клиент запрашивает несуществующий ресурс.

400 Bad Request – универсальный код ошибки, если серверу непонятен запрос от клиента.

403 Forbidden – возвращается, если операция запрещена для текущего пользователя.

415 Unsupported Media Type – возвращается, если фактический формат переданного содержимого не поддерживается.

418 I'm a Teapot – возвращается для неизвестных серверу запросов, которые не удалось даже разобрать.

500 Internal Server Error – возвращается, если на сервере вылетело необработанное исключение или произошла другая необработанная ошибка времени исполнения.

501 Not Implemented – Возвращается, если текущий метод неприменим (не реализован) к объекту запроса.

200 OK – возвращается, если запрос успешно обработан.

201 Created – возвращается, если объект успешно создан.

507 Insufficient Storage – возвращается, если место кончилось (хранилище переполнено).

Пример открытого API

Есть довольно много открытых API, работающих в формате REST. Большинство из них предоставляет доступ только для операций «чтения» объектов.

В качестве примера рассмотрим сайт Open Food Facts:
<https://world.openfoodfacts.org/data>

Данный сайт содержит информацию о продуктах. Ключом (id) служит бар-код продукта.

Пример



Часть 2

REST API СВОИМИ РУКАМИ

Веб-сервер

Для функционирования любого веб-приложения будь то API, информационная система или просто набор HTML-страниц (не в режиме доступа, как файл) нужен веб-сервер, который будет принимать HTTP-запросы и формировать на их основе HTTP-ответы.

Обычно используют промышленные веб-сервера, такие, как Apache или Nginx. Но нам для знакомства с построением собственного API будет достаточно чего-то более простого.

Фреймворк

Для создания современных веб-ориентированных информационных систем обычно используют фреймворки – готовые каркасы разработки, обеспечивающие поддержку определённых шаблонов проектирования (например, MVC).

Наиболее известны веб-фреймворки: Ruby on Rails, AngularJS, Django, Laravel и т.д.

Мы будем использовать «игрушечный» фреймворк для языка Python – Flask. В него встроен веб-сервер WSGI. Это позволит нам легко написать своё API, не вдаваясь в подробности веб-разработки.

Предварительная подготовка

1. Установим фреймворк: **python.exe -m pip install flask**
2. Создадим папку для приложения, например **FlaskPrj**
3. Запустим командную строку и перейдём в данную папку **cd FlaskPrj**
4. Запустим venv: **python.exe -m venv venv**
5. Активируем venv: **venv\Scripts\activate**

Первое веб-приложение на Flask

Напишем веб Hello World! Приложение, показывающее одну страницу с текстом «Hello World!»

```
from flask import Flask  
  
app = Flask(__name__)  
  
@app.route('/')  
def hello_world():  
    return 'Hello World!'
```

Как запустить?

Сначала нужно запустить веб-сервер. Это делается двумя командами в интерфейсе командной строки cmd:

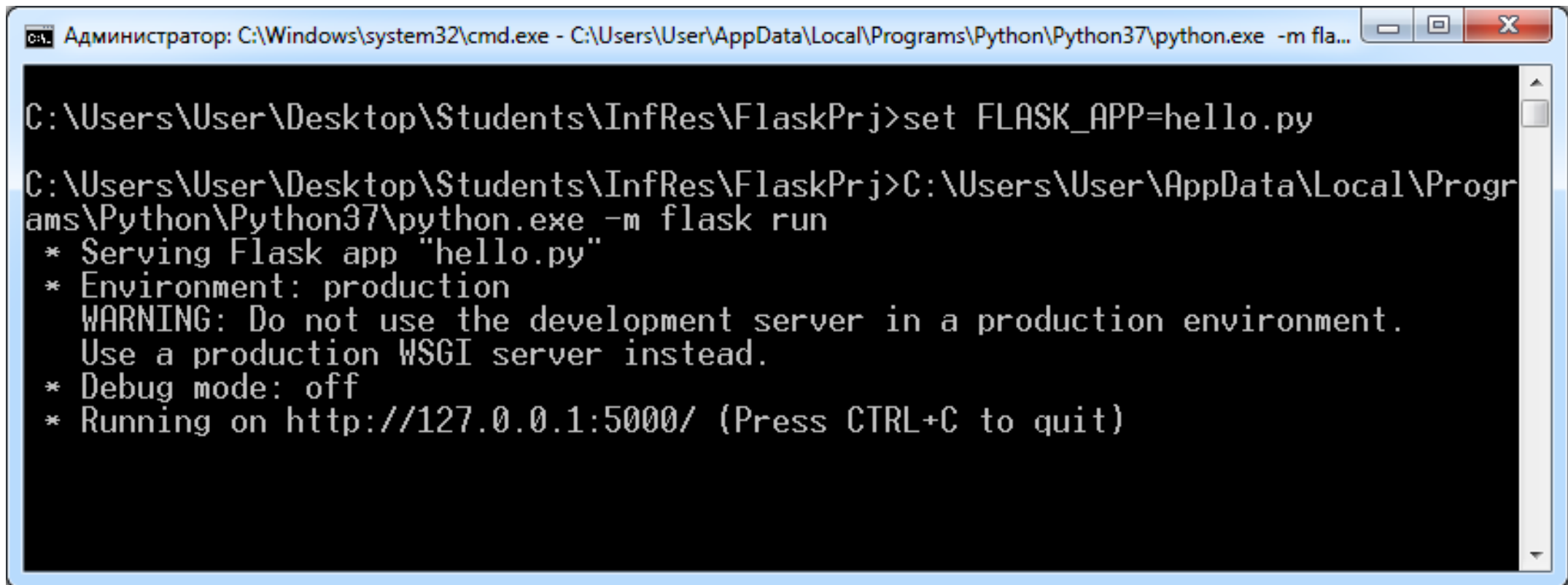
```
set FLASK_APP=hello.py
```

```
python.exe -m flask run
```

Затем запускаем браузер (например, firefox) и открываем локальную страницу на 5000 порту:

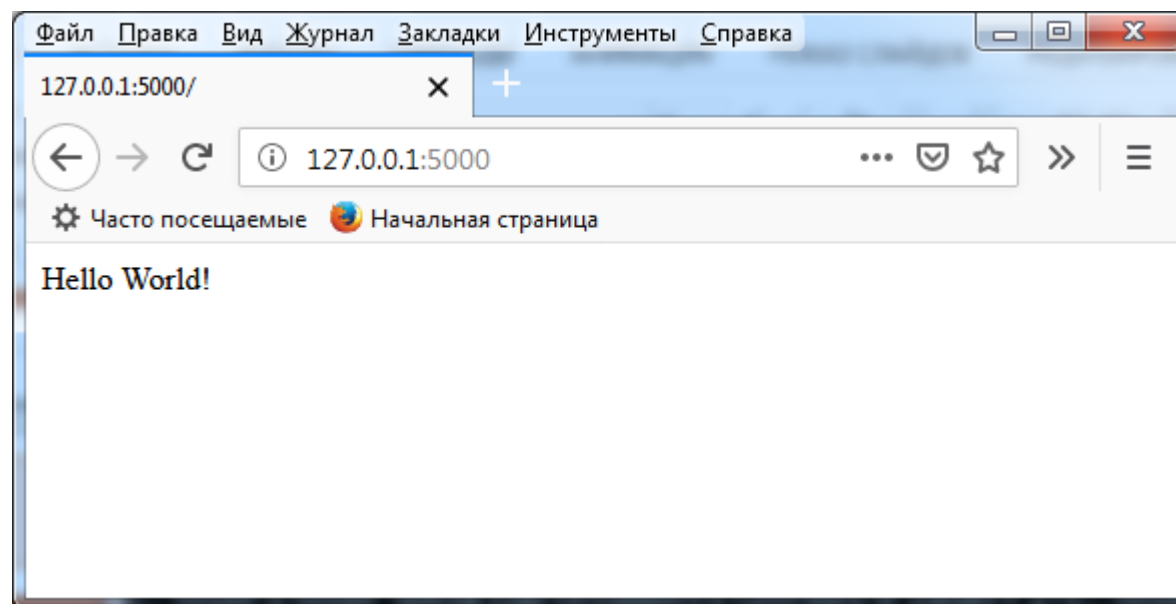
```
http://127.0.0.1:5000/
```

Запущенный сервер Flask

A screenshot of a Windows command prompt window. The title bar reads "Администратор: C:\Windows\system32\cmd.exe - C:\Users\User\AppData\Local\Programs\Python\Python37\python.exe -m fla...". The command prompt shows the following text:

```
C:\Users\User\Desktop\Students\InfRes\FlaskPrj>set FLASK_APP=hello.py  
C:\Users\User\Desktop\Students\InfRes\FlaskPrj>C:\Users\User\AppData\Local\Programs\Python\Python37\python.exe -m flask run  
* Serving Flask app "hello.py"  
* Environment: production  
  WARNING: Do not use the development server in a production environment.  
  Use a production WSGI server instead.  
* Debug mode: off  
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Результат



Route map веб-системы

Одним из способов атаки на веб-сайт может быть подбор непредназначенных для использования веб-адресов. Если веб-сервер настроен с минимальной защищённостью, то он будет обрабатывать каждое обращение и физически проверять, есть ли такой адрес. В случае наличия ошибок безопасности это может привести к скачиванию злоумышленников не предназначенных для него файлов, в том числе с частью исходного кода программы, а иногда, даже, к запуску непредназначенных для него скриптов.

Чтобы избежать подобных проблем все современные веб-системы допускают обращение только к адресам, входящим в карту адресов сайта (route map). Все обращения к адресам, не входящим в неё, даже, если такие адреса физически есть, игнорируются (возвращается ошибка 404 Not found).

Как устроена Flask-программа

from flask import Flask – подключение библиотеки Flask.

app = Flask(__name__) – запуск Flask с указанием названия файла, который реализует наше приложение.

def hello_world():

return 'Hello World!' – эти две строки описывают функцию, которая будет срабатывать при обращении к сайту. То, что она возвращает и будет веб-страницей.

@app.route('/') – это действие, написанное перед вызовом функции **hello_world()**, связывает вызов данной функции с адресом карты сайта '**адрес сайта/**'. К базовому адресу сайта дописывается аргумент **route**. В данном случае это просто слеш.

```
from flask import Flask  
app = Flask(__name__)
```

```
@app.route('/')  
def hello_world():  
    return 'Hello World!'
```


Как описывается Route map в Flask

Чтобы сопоставить определённой функции некоторый веб-адрес используется функция-декоратор `route`. Первый аргумент – это сопоставляемый адрес без базового URL:

```
@app.route('/')
```

```
@app.route('/hello')
```

Кроме того, может быть указан список HTTP-методов, с помощью которых возможно обращение к данному адресу. Для этого используется конструкция `methods = []`

```
@app.route('/hello', methods=['POST'])
```

```
@app.route('/', methods=['GET', 'POST', 'PUT'])
```

Как описывается Route map в Flask

Также в адрес могут быть встроены значения параметров. Тогда в адресной карте сайта может быть передана информация как о названии параметра, так и о его типе данных. Flask маршрутизация допускает 5 типов данных:

- строка `string` – любой текст без слешей (по умолчанию);
- целое число `int` – любые целые числа;
- вещественные числа `float` – любые вещественные числа, разделённые точкой;
- путь `path` – любые строки, включая слеш;
- `uuid` – строки формата UUID.

Пример:

```
@app.route('/user/<username>')
```

```
@app.route('/user/<int:user_id>')
```

```
@app.route('/user/<path:page>')
```

Принятые названия функций REST

- **index** – получение списка объектов данного типа
- **show_** – получение одного объекта данного типа.
- **create** – создание одного нового объекта данного типа
- **update** – обновление, модификация одного существующего объекта данного типа.
- **destroy** – удаление одного существующего объекта данного типа.

Підготуємо функції для REST

```
from flask import Flask

app = Flask(__name__)

@app.route('/api/rows', methods=['GET'])
def rows_index():
    return 'Many rows!'

@app.route('/api/rows/<int:row_id>', methods=['GET'])
def rows_show(row_id):
    return 'One row!'
```

```
@app.route('/api/rows', methods=['POST'])
def rows_create():
    return 'New row!'

@app.route('/api/rows/<int:row_id>', methods=['PUT'])
def rows_update(row_id):
    return 'Update row!'

@app.route('/api/rows/<int:row_id>', methods=['DELETE'])
def rows_destroy(row_id):
    return 'Delete row!'
```

Что дальше?

Теперь, когда у нас есть каркас нужно реализовывать само содержимое методов. Для этого нам нужны данные. Обычно информация хранится либо в базе данных, либо в файле, либо ещё в каком-либо хранилище данных. Но мы для простоты рассмотрим ситуацию, когда информация – это всего лишь Python список.

Также подключим две дополнительных библиотеки: `request` и `jsonify` для получения GET или POST-параметров HTTP-запроса и для преобразования результата в формат JSON.

Наши данные

```
from flask import request
```

```
from flask import jsonify
```

```
rows_base = [
```

```
{'id': 1, 'info': 'Тест 1'},
```

```
{'id': 2, 'info': 'Тест 2'},
```

```
{'id': 3, 'info': 'Тест 3'},
```

```
{'id': 4, 'info': 'Тест 4'},
```

```
{'id': 5, 'info': 'Тест 5'}
```

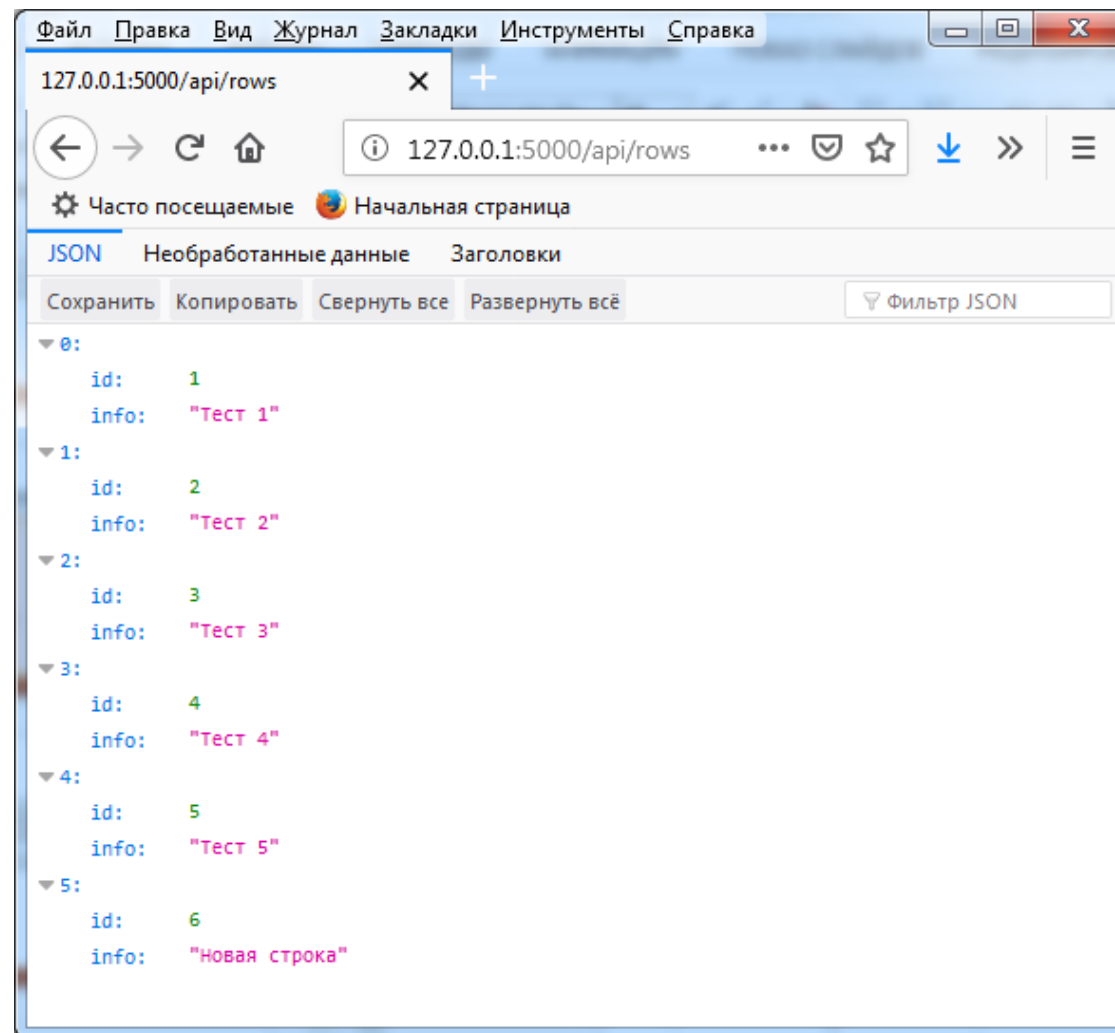
```
]
```

```
rows_id_seq = 6
```

Реализуем список объектов

```
@app.route('/api/rows', methods=['GET'])  
def rows_index():  
    return jsonify(rows_base)
```

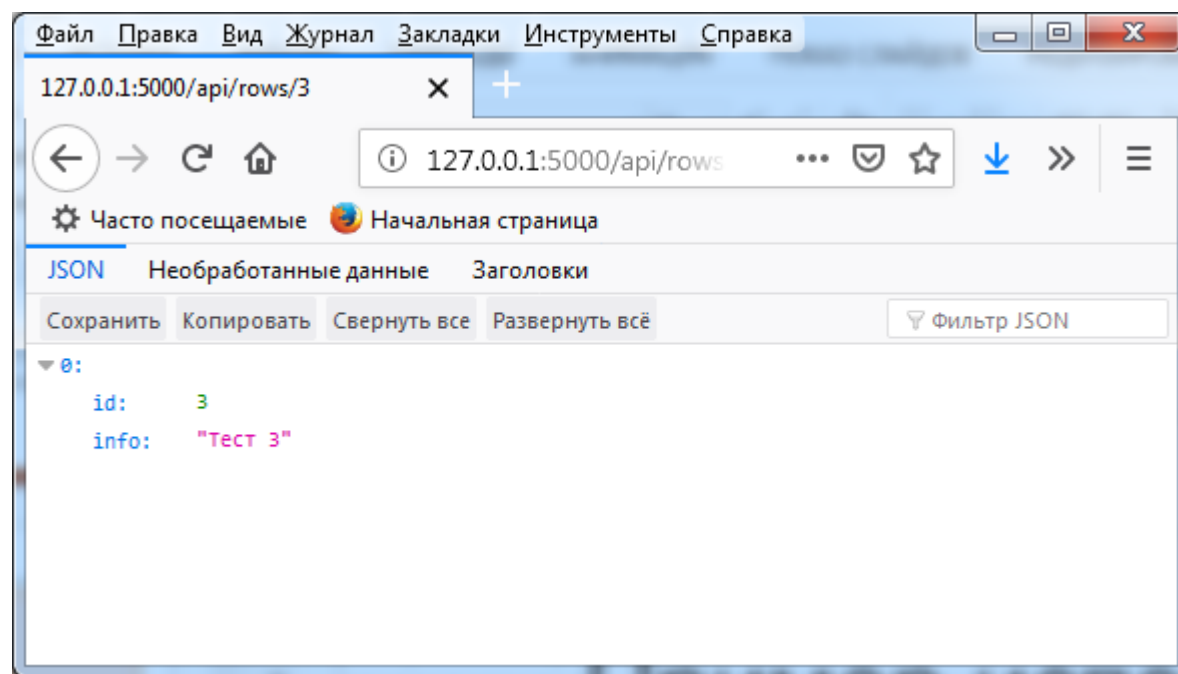
Пример использования



Реализуем отображение одного объекта

```
@app.route('/api/rows/<int:row_id>', methods=['GET'])  
def rows_show(row_id):  
    return jsonify([a for a in rows_base if a['id'] == row_id])
```

Пример использования



Реализуем добавление объекта

```
@app.route('/api/rows', methods=['POST'])
def rows_create():
    global rows_id_seq
    info = request.form['info']
    rows_base.append({'id':rows_id_seq,'info':info})
    x = rows_base[len(rows_base) - 1]
    rows_id_seq += 1
    return jsonify(x)
```

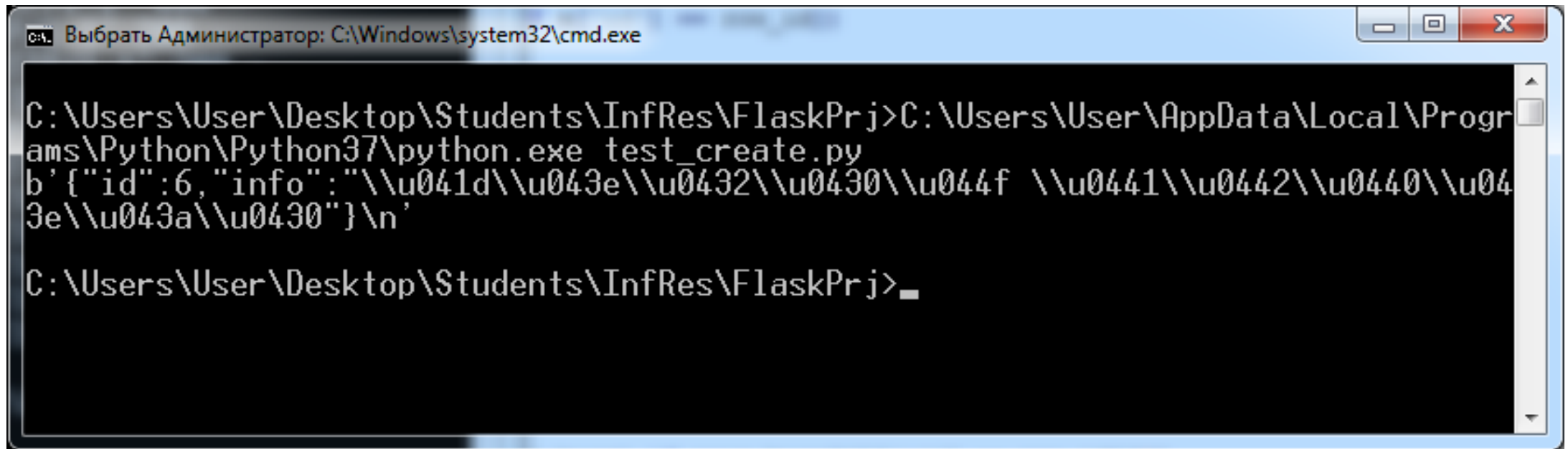
Просто так это уже не запустишь!

```
import urllib.request
import urllib.parse

url = 'http://127.0.0.1:5000/api/rows'
values = {
    'info': 'Новая строка'
}

data = urllib.parse.urlencode(values).encode()
req = urllib.request.Request(url, data)
page_string = urllib.request.urlopen(req).read()
print(page_string)
```

Результат



The screenshot shows a Windows Command Prompt window titled "Выбрать Администратор: C:\Windows\system32\cmd.exe". The prompt is at the directory C:\Users\User\Desktop\Students\InfRes\FlaskPrj. The user has executed the command C:\Users\User\AppData\Local\Programs\Python\Python37\python.exe test_create.py. The output is a JSON string: b'{"id":6,"info":"\\u041d\\u043e\\u0432\\u0430\\u0440\\u0441\\u0430\\u0430"}\n'. The prompt is now at C:\Users\User\Desktop\Students\InfRes\FlaskPrj>.

```
C:\Users\User\Desktop\Students\InfRes\FlaskPrj>C:\Users\User\AppData\Local\Programs\Python\Python37\python.exe test_create.py
b'{"id":6,"info":"\\u041d\\u043e\\u0432\\u0430\\u0440\\u0441\\u0430\\u0430"}\n'
C:\Users\User\Desktop\Students\InfRes\FlaskPrj>_
```

Реализуем редактирование объекта

```
@app.route('/api/rows/<int:row_id>', methods=['PUT'])
def rows_update(row_id):
    info = request.form['info']
    row_num = [i for i, _ in enumerate(rows_base) if rows_base[i]['id'] == row_id][0]
    rows_base[row_num] = {'id':row_id,'info':info}
    return jsonify(rows_base[row_num])
```

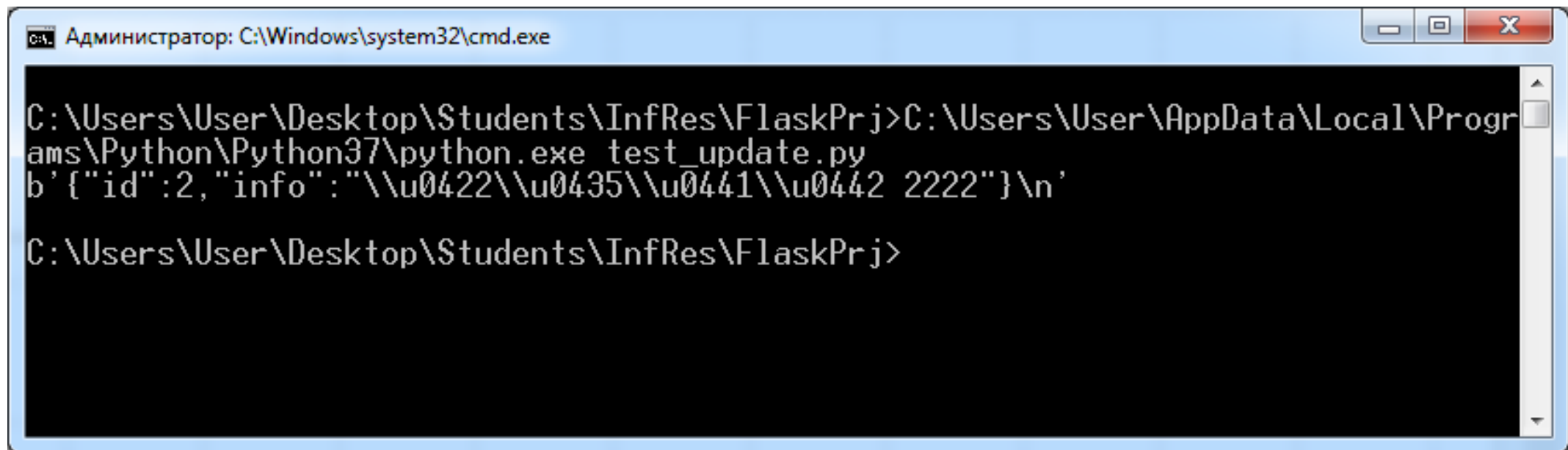
Запуск

```
import urllib.request
import urllib.parse

url = 'http://127.0.0.1:5000/api/rows/2'
values = {
    'info':    'Тест 2222'
}

data = urllib.parse.urlencode(values).encode()
req = urllib.request.Request(url = url, data = data, method='PUT')
page_string = urllib.request.urlopen(req).read()
print(page_string)
```

Результат



Администратор: C:\Windows\system32\cmd.exe

```
C:\Users\User\Desktop\Students\InfRes\FlaskPrj>C:\Users\User\AppData\Local\Programs\Python\Python37\python.exe test_update.py  
b'{"id":2,"info":"\\u0422\\u0435\\u0441\\u0442 2222"}\n'  
C:\Users\User\Desktop\Students\InfRes\FlaskPrj>
```


Реализуем удаление объекта

```
@app.route('/api/rows/<int:row_id>', methods=['DELETE'])
def rows_destroy(row_id):
    row_num = [i for i, _ in enumerate(rows_base) if rows_base[i]['id'] == row_id][0]
    x = rows_base[row_num]
    rows_base.pop(row_num)
    return jsonify(x)
```

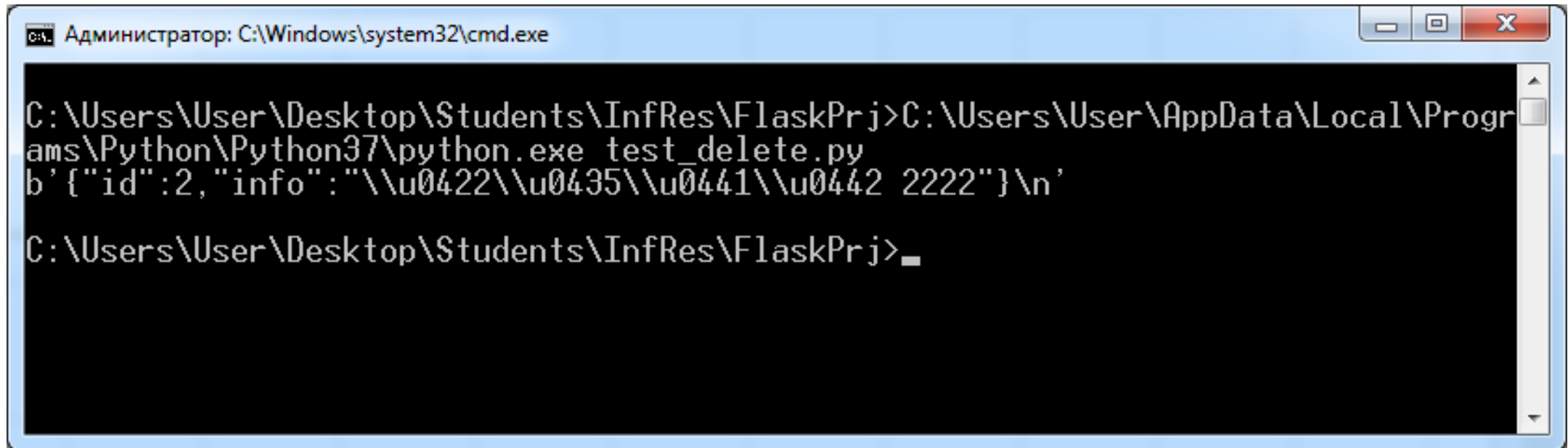
Запуск

```
import urllib.request
import urllib.parse

url = 'http://127.0.0.1:5000/api/rows/2'

req = urllib.request.Request(url = url, method='DELETE')
page_string = urllib.request.urlopen(req).read()
print(page_string)
```

Результат



Администратор: C:\Windows\system32\cmd.exe

```
C:\Users\User\Desktop\Students\InfRes\FlaskPrj>C:\Users\User\AppData\Local\Programs\Python\Python37\python.exe test_delete.py
b'{"id":2,"info":"\\u0422\\u0435\\u0441\\u0442 2222"}\n'

C:\Users\User\Desktop\Students\InfRes\FlaskPrj>_
```

Полезные ссылки

1. <http://www.json.org/> – Сайт формата JSON.
2. <https://apptractor.ru/info/articles/10-rest-api.html>
3. <https://palletsprojects.com/p/flask/>
4. <http://flask.pocoo.org/docs/1.0/quickstart/#routing>
5. <https://www.kaggle.com/sohier/beyond-queries-exploring-the-bigquery-api>
6. <https://googleapis.dev/python/bigquery/latest/index.html>