нетология

Flask

КИРИЛЛ ТАБЕЛЬСКИЙ

КИРИЛЛ ТАБЕЛЬСКИЙ

Lightmap

План занятия

1. Что такое Flask
2. Простейшее приложение
3. Работа с БД
4. Дополнительные материалы
5. Домашнее задание

Что такое Flask

Flask — это микрофреймворк, предназначенный для создания легковесного http-сервера.

«Микро» означает, что простое приложение может поместиться в один файл и далее его можно развивать, постепенно добавляя функционал

Flask vs Django

Плюсы Flask:

- гибкость;
- скорость;
- мало магии.

Плюсы Django:

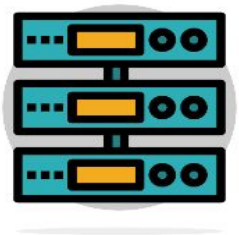
- все включено;
- скорость разработки.

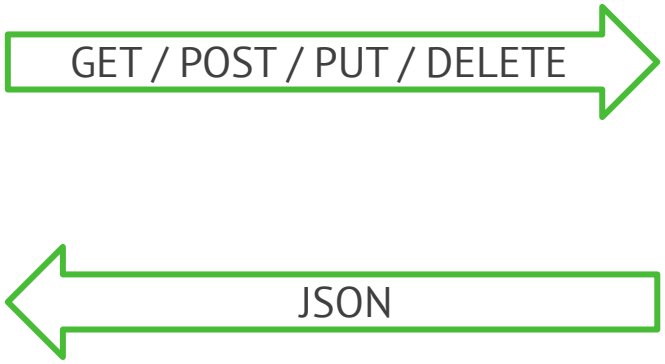
django

Сценарии использования Flask

- Микросервисы
- Небольшие API
- MOCK API

REST API





REST API

HTTP методы

Метод	Действие
GET	Получить данные
POST	Вставить данные в базу
PUT	Полность изменить запись в базе
PATCH	Частично изменить запись в базе
DELETE	Удалить запись из базы

HTTP статусы

Статус	Описание
200	OK
400	Неправильный формат запроса
401	Ошибка авторизации
404	Нет такой записи в базе
500	Внутренняя ошибка

Из чего состоит запрос

- uri (query string);
- body
- headers
- cookies

```
requests.post(
    'http://home.come/some_resource/42?key_1=value_1&key_2=value_2',
    json={
        'hello': 'world'
    },
    headers={
        'token': 'some-token'
    }
)
```

Минимальное приложение

```
from flask import Flask



app = Flask('app')



if __name__ == '__main__':
    app.run(host='127.0.0.1', port=5000)
```

Простейшее вью

```
from flask import Flask
from flask import jsonify


app = Flask('app')


def hello_world():
    return jsonify({'hello': 'world'})


app.add_url_rule('/hello_world/', view_func=hello_world, methods=['GET'])

if __name__ == '__main__':
    app.run(host='127.0.0.1', port=5000)
```

Class based вью

```
from flask.views import MethodView
…
class HelloWorld(MethodView):

    def get(self):
        return jsonify({'http_method': 'GET', 'hello': 'world'})
    def post(self):
        return jsonify({'http_method': 'POST', 'hello': 'world'})
    def patch(self):
        return jsonify({'http_method': 'PATCH', 'hello': 'world'})
    def delete(self):
        return jsonify({'http_method': 'DELETE', 'hello': 'world'})

app.add_url_rule( '/hello_world/',
view_func=HelloWorld.as_view('hello_world'),
methods=['GET', 'POST', 'PATCH', 'DELETE']
)
```

Переменная в URL

```
class HelloWorld(MethodView):


    def get(self, some_variable: int):
        …


app.add_url_rule(
    '/hello_world/<int:some_variable>/',
    view_func=HelloWorld.as_view('hello_world'),
    methods=['GET', 'POST', 'PATCH', 'DELETE']
)
```

```
requests.get('/hello_world/<int:some_variable>/')
```

Чтение json

```
from flask import request


class HelloWorld(MethodView):
    def post(self):
        json_data = request.json
        print(json_data)

        ...
        return jsonify(...)
```

```
requests.post('...', json={'hello': 'world'})
```

Чтение заголовков

```
class HelloWorld(MethodView):


    def post(self):
        token = request.headers.get('token')
        print(token)


        ...
        return jsonify(...)
```

```
data = requests.post('...', headers={'token': 'some_token'})
```

Чтение QS

```
class HelloWorld(MethodView):


    def post(self):
        query_string_data = request.args
        value_1 = query_string_data.get('key_1')


        ...
        return jsonify(...)
```

```
data = requests.post(
'http://127.0.0.1:5000/hello_world/?key_1=value_1&key_2=value_2 '
```

Обработка ошибок

```
app = Flask('app')


class HttpError(Exception):


    def __init__(self, status_code: int, message: str | dict |
list):
        self.status_code = status_code
        self.message = message


@app.errorhandler(HttpError)
def error_handler(error: HttpError):
    response = jsonify({'status': 'error',
'message':error.message})
    response.status_code = error.status_code
    return response
```

Обработка ошибок

```
class SomeResourceView(MethodView):


    def get(self):
        ...
        if (...):
            raise HttpError(404, 'resource not found')


        return jsonify({...})
```

Схема валидации

```
import pydantic
from typing import Type, Optional


class SomeResourceValidator(pydantic.BaseModel):


    string_field: str
    int_field: int
    small_string_field: str
    optional_int: Optional[int]


    @pydantic.validator('small_string_field')
    def check_small_string(cls, value: str):
        if len(value) > 4:
            raise ValueError('length of small_string_field must be less
than 4')
        return value
```

Валидация

```
def validate_resource(
        data: dict,
        model_cls: Type[SomeResourceValidator]
):
    try:
        model = model_cls(**data)
        return model.dict(exclude_none=True)
    except pydantic.ValidationError as error:
        raise HttpError(400, error.errors())
```

Валидация

```
class SomeResourceView(MethodView):



    def post(self):
        validated_json = validate_resource(request.json, SomeResourceValidator)
        print(validated_json)
        ...
        return jsonify({})
```

```
data = requests.post('http://127.0.0.1:5000/some_resource/', json={
    'string_field': 'hello world',
    'int_field': 4,
    'small_string_field': 'a'
})
```

Работа с БД

Во Flask нет ORM.

Используем:

1. SQL Alchemy
2. PonyORM
3. Голый язык запросов
4. **Не забываем хешировать пароли**

Запуск продакт сервера

```
pip install gunicorn
```

```
#app.py
from flask import Flask

app = Flask('app')
```

```
gunicorn -b 0.0.0.0:5000 app:app --capture-output
```

Дополнительные материалы

- https://flask.palletsprojects.com
- https://blog.miguelgrinberg.com/post/restful-authentication-with-flask

  — примеры авторизации

Домашнее задание

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задаём в чате вашей группы!
- Задачи можно сдавать по частям.
- Зачёт по домашней работе проставляется после того, как приняты **все задачи**.

нетология

Задавайте вопросы и напишите отзыв о лекции!

КИРИЛЛ ТАБЕЛЬСКИЙ