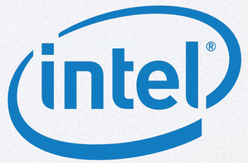
BUILDING BEAUTIFUL **REST** APIs

with Flask, Swagger UI and Flask-RESTPlus

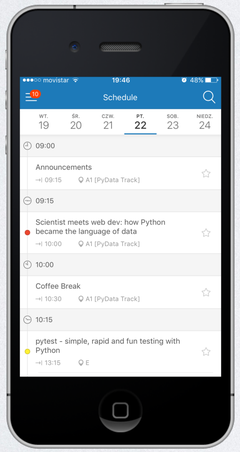
Michał Karzyński • EuroPython 2016

ABOUT ME

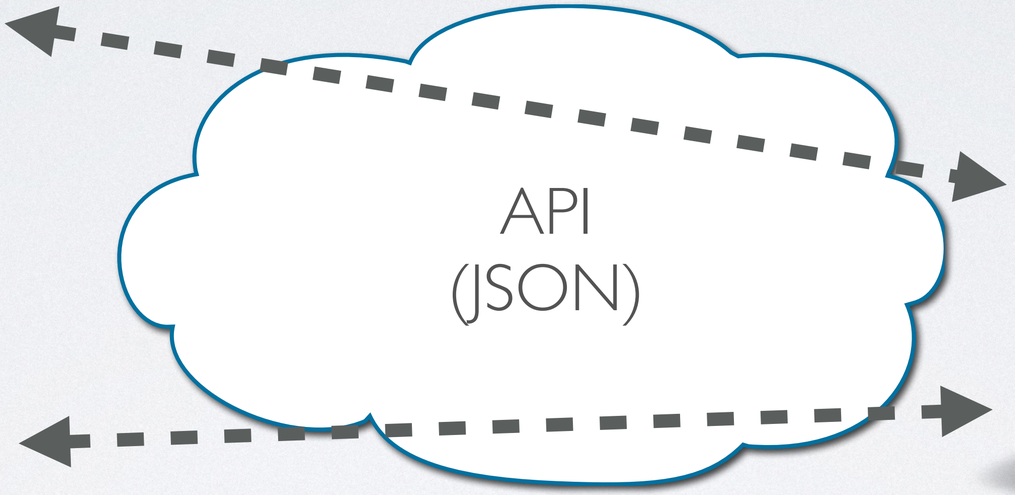
- My name is Michał Karzyński (that's Polish for Mike)
- I blog at http://michal.karzynski.pl
  Short URL: karzyn.com
- I wrote a book for Linux admins, I write code in Python and JavaScript
- I'm the tech lead of a Web UI team at 

WHAT IS A WEB **API**?

Web (JavaScript)



Phone (Swift, Java)



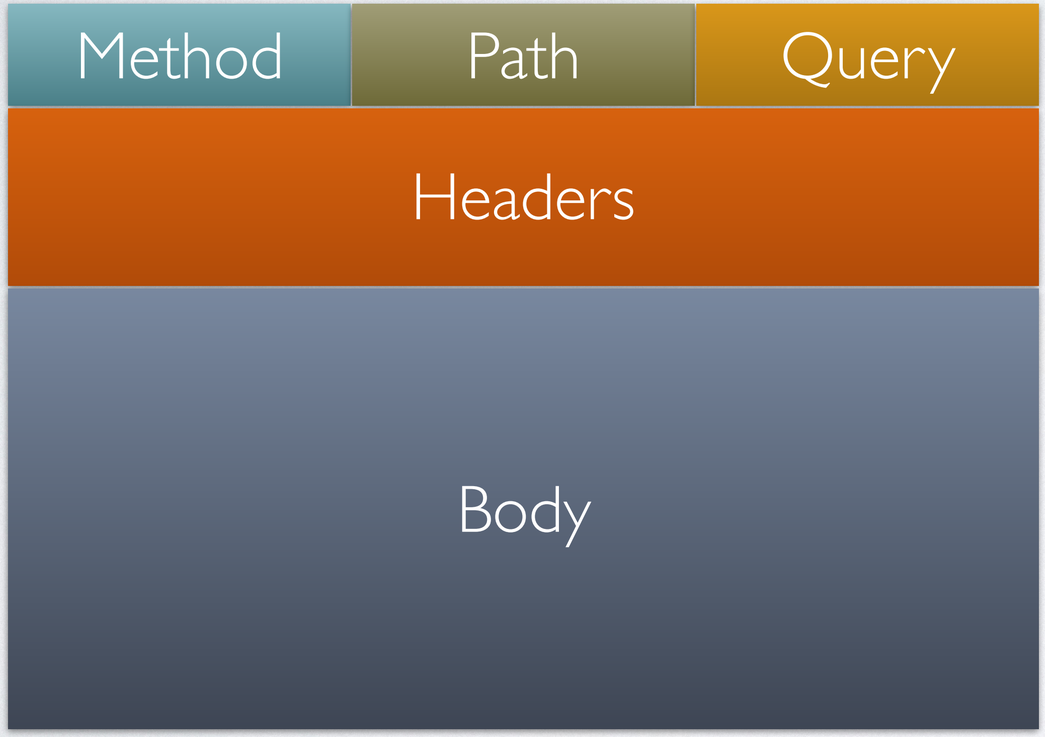
Server
(Python)

WHAT IS A **REST** API?

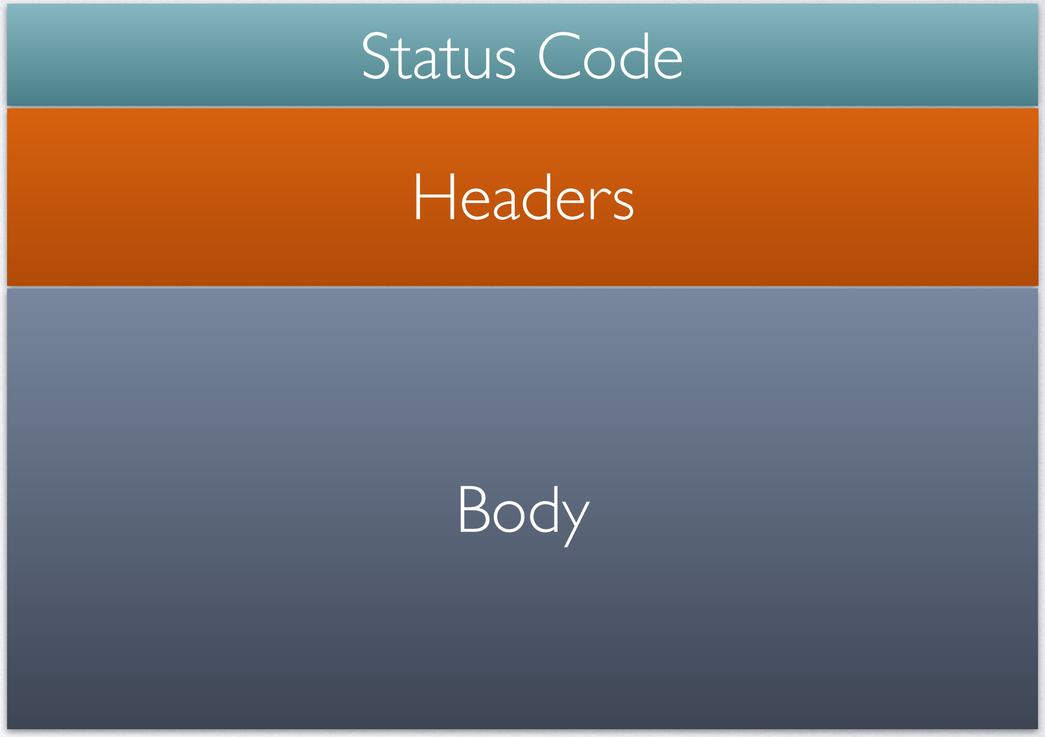
REPRESENTATIONAL STATE TRANSFER

A clever way to use **HTTP** to build APIs.

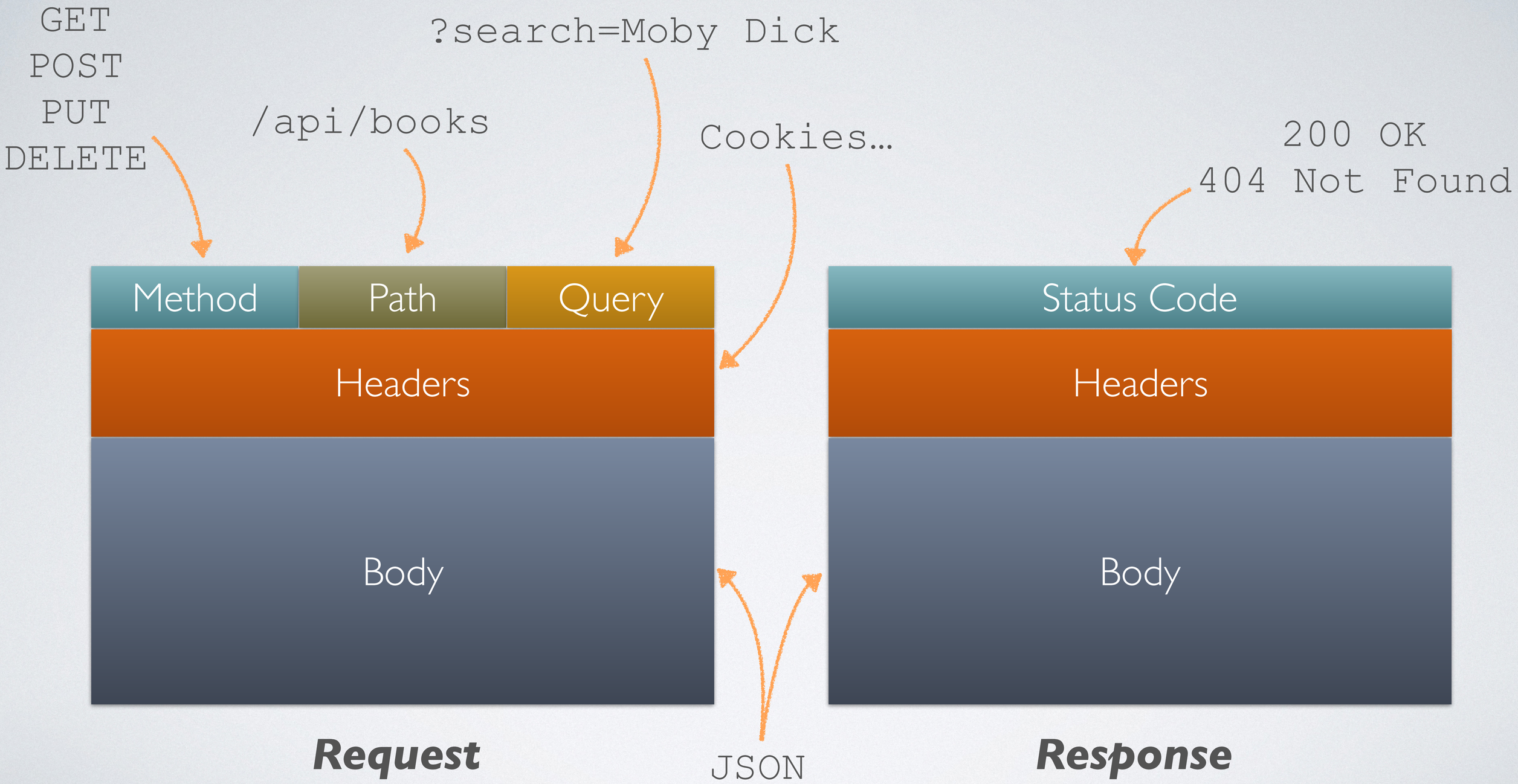
ANATOMY OF **HTTP**



Request



Response


GET
POST
PUT
DELETE
/api/books
?search=Moby Dick
Cookies…
200 OK
404 Not Found
Method
Path
Query
Headers
Body
Status Code
Headers
Body
Request
JSON
Response

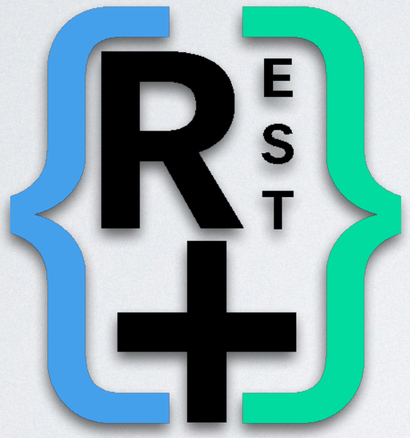
REST CONVENTIONS

	GET	PUT	POST	DELETE
Collection `/books`	List books		New book	
Item `/books/123`	Display book	Update book		Delete book
Controller `/books/123/borrow`			Borrow book	

FLASK

flask.pocoo.org

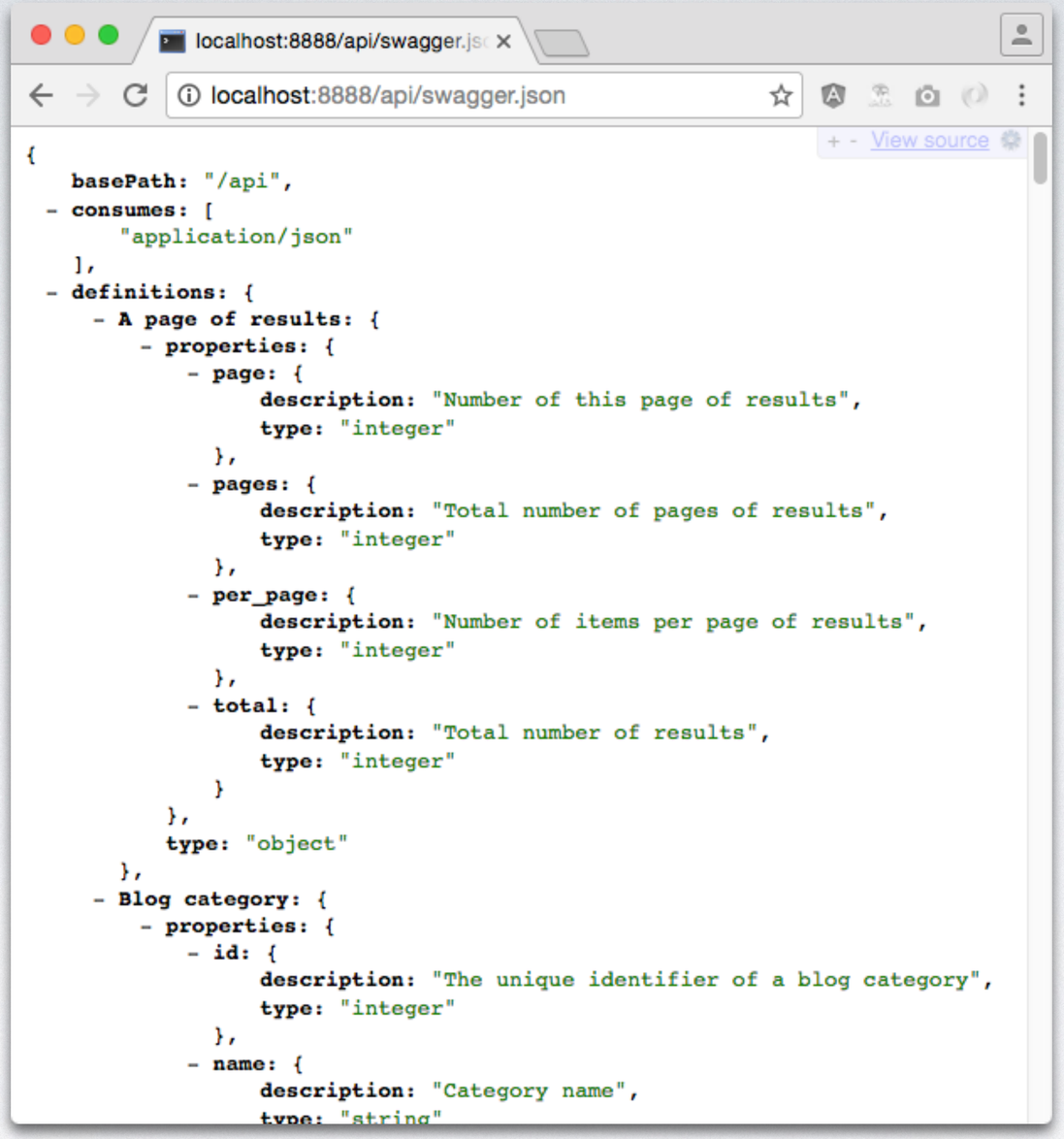
FLASK-RESTPlus



flask-restplus.rtfd.io

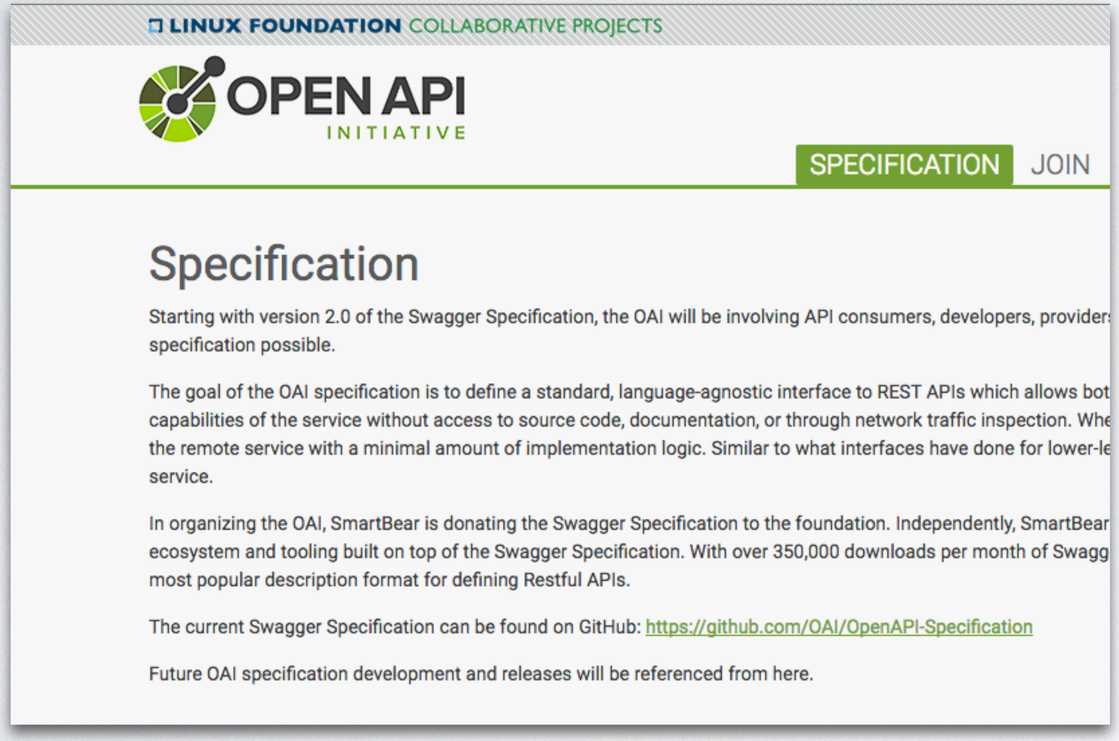
- define and document endpoints
- validate input
- format output (as JSON)
- turn Python exceptions into HTTP responses
- minimise boilerplate code
- generate interactive documentation (**Swagger UI**)

Demo



OPEN API FORMAT

OPEN API SPECIFICATION

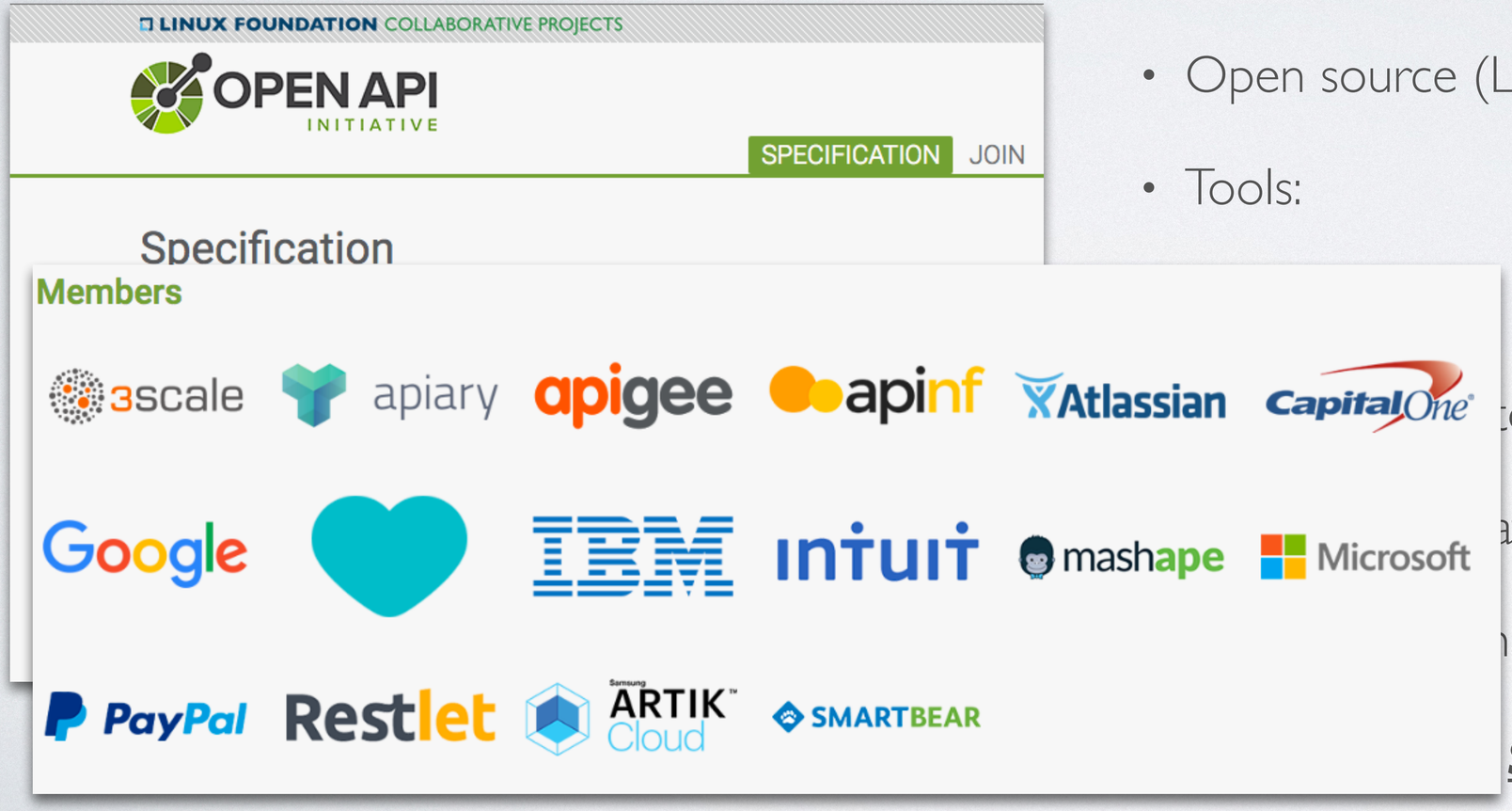


- Standard language to describe REST APIs
- Open source (Linux Foundation)
- Tools:
  - Swagger UI
  - Swagger Editor
  - Code generators
- Initiative with many powerful members

openapis.org

swagger.io

OPEN API SPECIFICATION



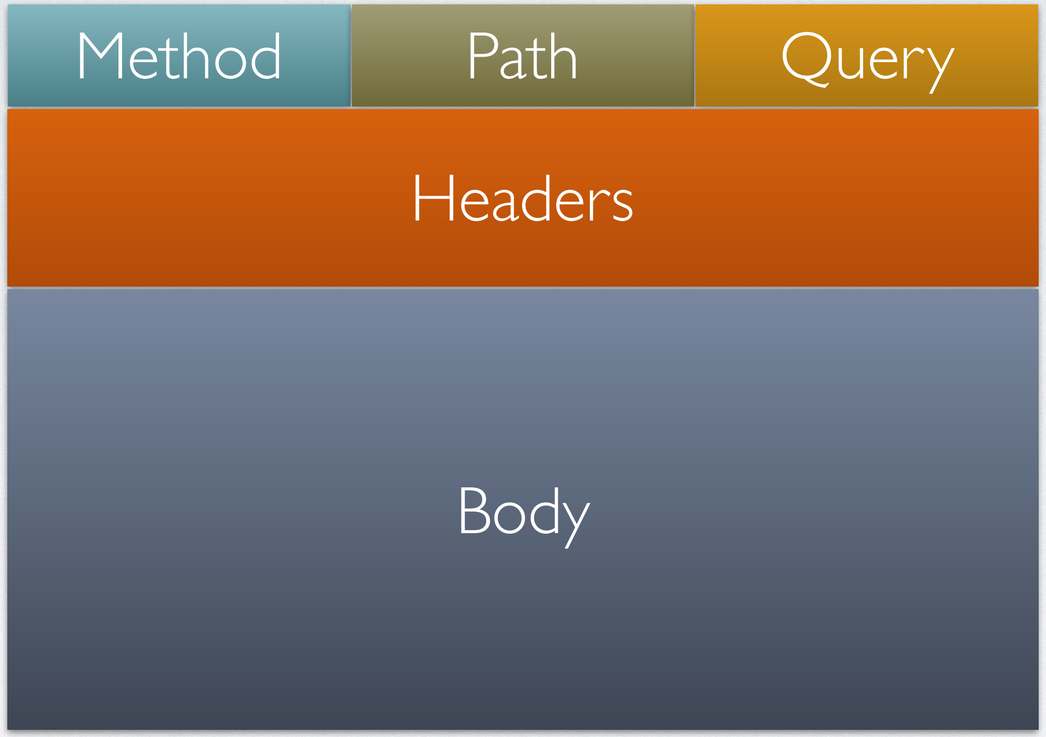
- Standard language to describe REST APIs
- Open source (Linux Foundation)
- Tools:

tor

ators

nany powerful members

swagger.io



Request

REQUEST METHOD

POST /api/books/123/borrow?when=today


```python
from flask_restplus import Resource

@api.route('/<int:id>/borrow')
class BorrowBookController(Resource):

    def post(self, id):
        """ Borrow book from library.

        Allows the current user to borrow
        the book out of the library.
        """

        ...
        return {'message': 'OK'}
```

REQUEST METHOD

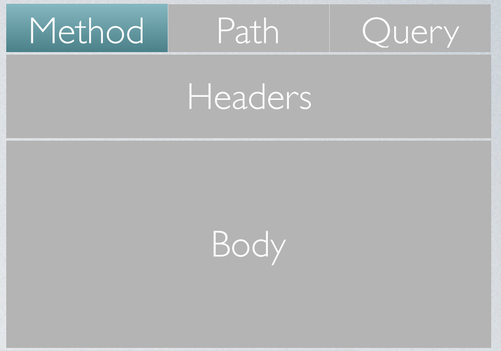
POST /api/books/123/borrow?when=today


```python
from flask_restplus import Resource

@api.route('/<int:id>/borrow')
class BorrowBookController(Resource):

    def post(self, id):
        """ Borrow book from library.

        Allows the current user to borro
        the book out of the library.
        """
        ...
        return {'message': 'OK'}
```

```python
class Resource:

    def get(self)...
    def post(self)...
    def put(self)...
    def delete(self)...
    def patch(self)...
    def options(self)...
    def head(self)...
```

REQUEST METHOD

POST /api/books/123/borrow?when=today



POST /books/{id}/borrow — Borrow book from library

Implementation Notes

Allows the current user to borrow the book out of the library.

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	(required)		path	integer

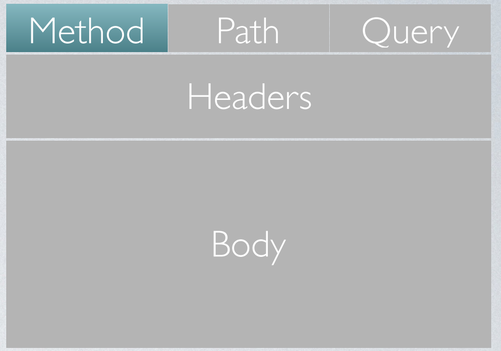
Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	Success		

Try it out!

REQUEST METHOD

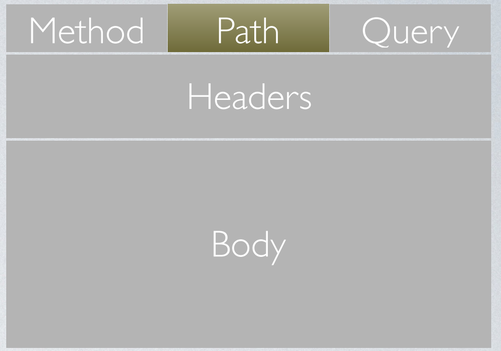
POST `/api/books/123/borrow?when=today`



GET	/blog/categories/	Returns list of blog categories
POST	/blog/categories/	Creates a new blog category
DELETE	/blog/categories/{id}	Deletes blog category
GET	/blog/categories/{id}	Returns a category with a list of posts
PUT	/blog/categories/{id}	Updates a blog category

REQUEST PATH

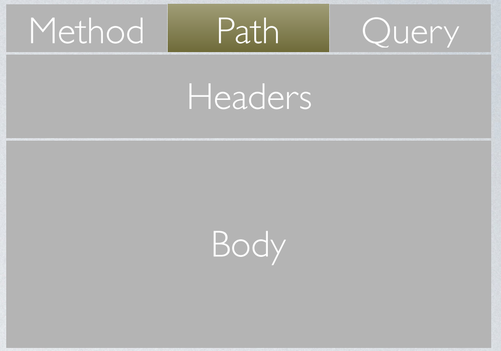
POST **/api/books/123/borrow**?when=today




```
@api.route(‘/books/<int:id>/borrow’)

@api.route('/articles/<title>')

@api.route('/wiki/<path:wikipage>')

@api.route('/values/<float:value>')

@api.route('/object/<uuid:identifier>')
```

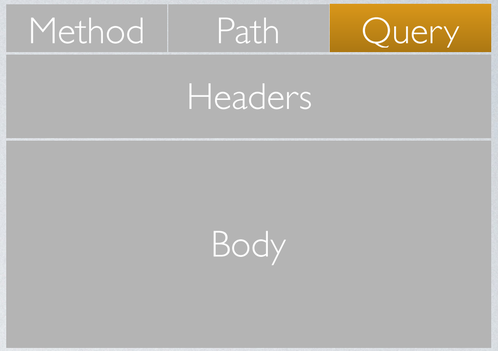
REQUEST PATH

`GET /api/book/123/borrow?when=today`



Parameters

Parameter	Value	Description	Parameter Type	Data Type
wikipage	(required)		path	string
title	(required)		path	string
identifier	(required)		path	string
id	(required)		path	integer
value	(required)		path	double



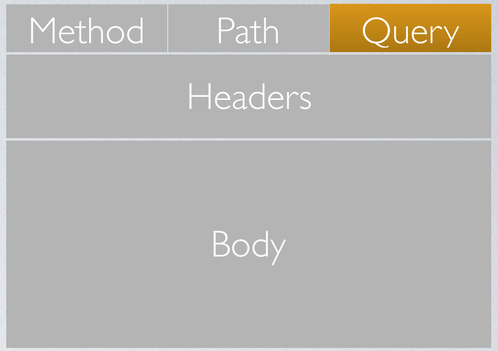
QUERY ARGUMENTS

`GET /api/books?page=1&per_page=10`

```
from flask_restplus import reqparse

pagination = reqparse.RequestParser()
pagination.add_argument('page', type=int, required=False,
                        default=1, help='Page number')
pagination.add_argument('per_page', type=int, required=False,
                        choices=[10, 20, 30, 40, 50])
```

QUERY ARGUMENTS

`GET /api/books?page=1&per_page=10`




```
from flask import request
from flask_restplus import Resource

@api.route('/')
class PostsCollection(Resource):

    @api.expect(parsers.pagination)
    def get(self):
        args = pagination_arguments.parse_args(request)
        page = args.get('page', 1)
        per_page = args.get('per_page', 10)
        ...
```

Method	Path	Query
Headers		
Body		

QUERY ARGUMENTS

`GET /api/books?page=1&per_page=10`

Parameters

Parameter	Value	Description	Parameter Type	Data Type
per_page	10 (default)	Results per page	query	integer
page	1	Page number	query	integer

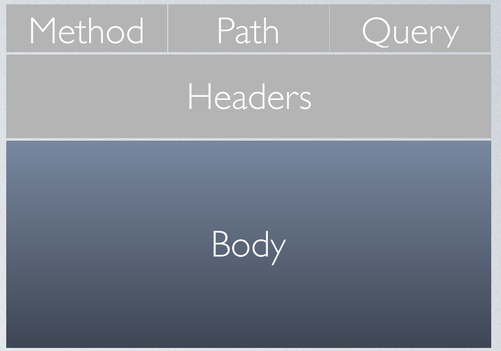
Try it out!

REQUEST BODY (JSON)
API MODELS


```
blog_post = api.model('Blog post', {
    'title': fields.String(description='Article title'),
    'body': fields.String(description='Article content'),
    'pub_date': fields.DateTime,
    'category_id': fields.Integer(min=1),
})
```

```
@api.expect(blog_post)
def post(self):
    ...
```

REQUEST BODY (JSON)
API MODELS



Parameters

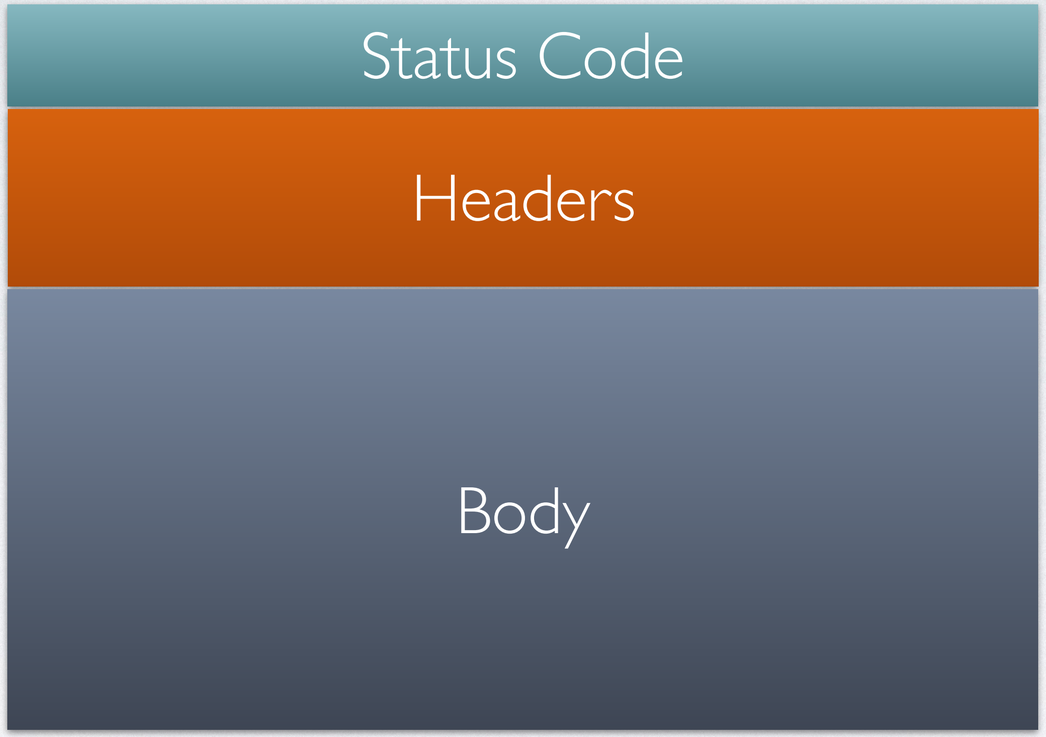
Parameter	Value	Parameter Type	Data Type
payload	(required)	body	Model | Model Schema

Parameter content type: application/json

Blog post {
body (string): Article content,
category_id (integer, *optional*),
pub_date (string, *optional*),
title (string): Article title
}

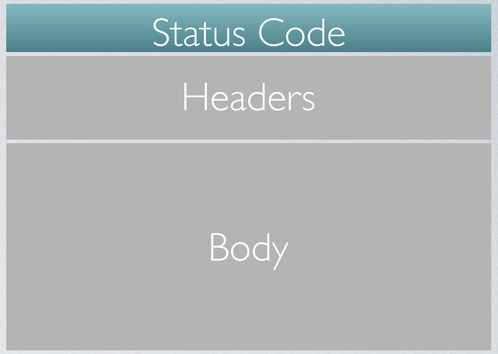
API MODELS
INHERITANCE AND **NESTING**


```
category = api.model('Blog category', {
    'id': fields.Integer(description='The unique id of category'),
    'name': fields.String(description='Category name'),
})

category_with_posts = api.inherit('Blog category with posts', category,
{
    'posts': fields.List(fields.Nested(blog_post))
})
```



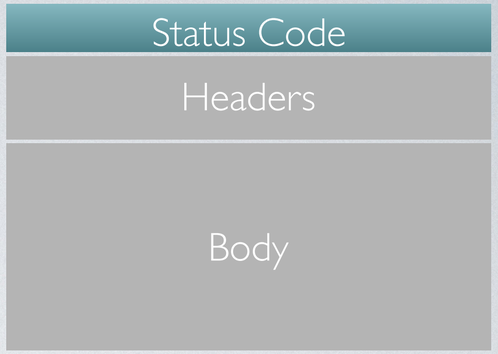
Response

RESPONSE STATUS CODE




```python
@api.route('/<int:id>')
@api.response(404, 'Post not found.')
class PostItem(Resource):

    @api.response(204, 'Post successfully deleted.')
    def delete(self, id):
        """
        Deletes blog post.
        """
        delete_post(id)
        return None, 204
```

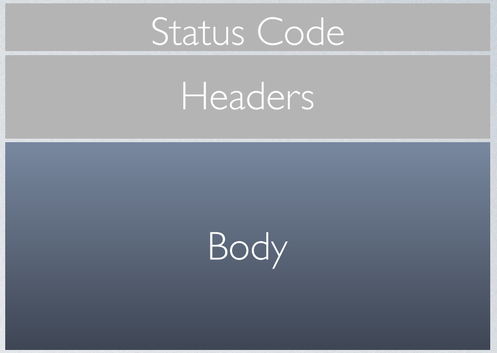
RESPONSE STATUS CODE



Response Messages

HTTP Status Code	Reason
204	Post successfully deleted.
404	Post not found.

RESPONSE BODY (JSON)




```
blog_post = api.model('Blog post', {
    ...
    'category_id': fields.Integer(attribute='category.id'),

    'name': fields.String(attribute=lambda x: x._private_name),
})
```

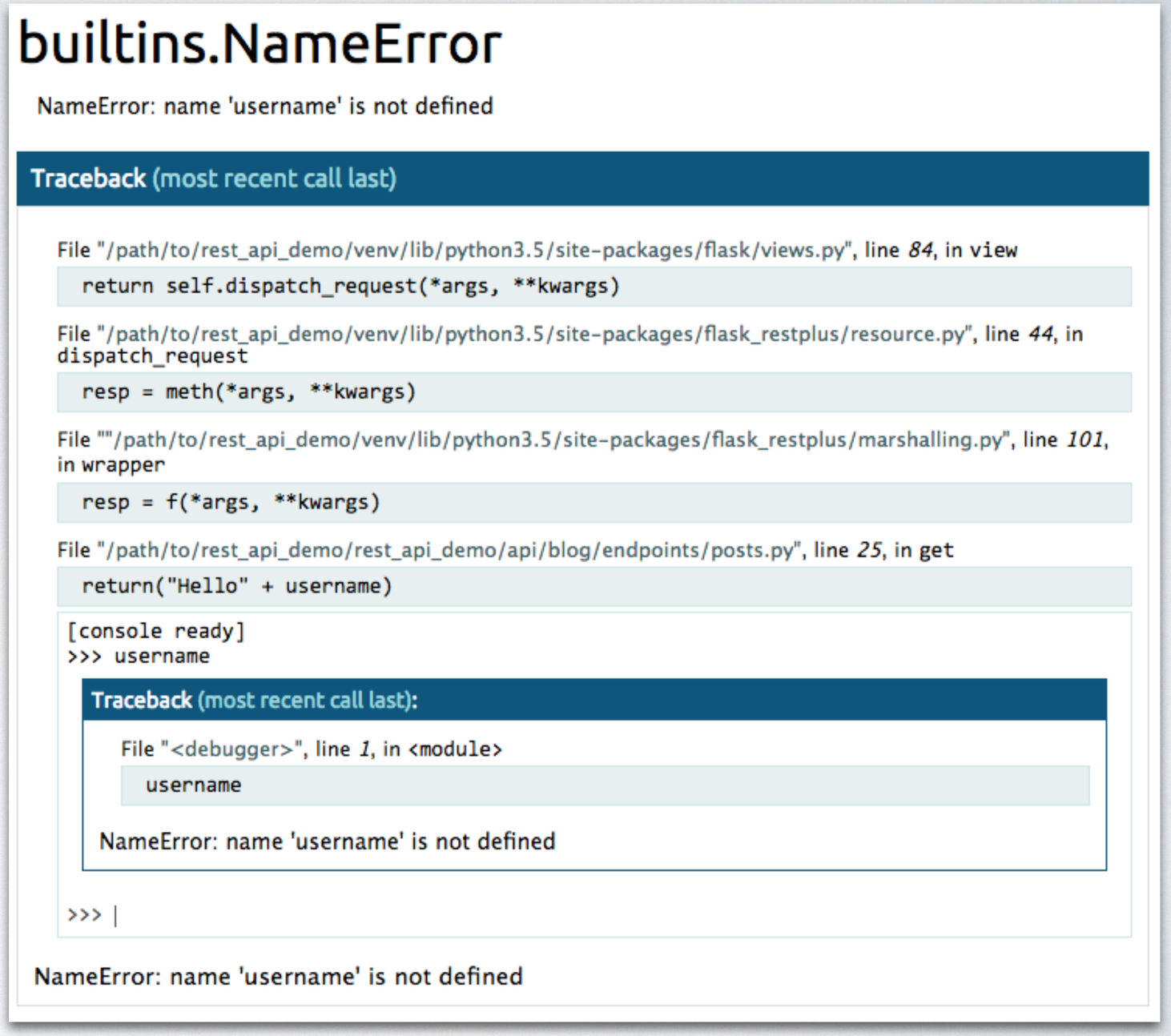
```
@api.marshal_with(blog_post)
def get(self):
    ...
```

```
@api.marshal_list_with(blog_post)
def get(self):
    ...
```

EXCEPTION HANDLING

```
from sqlalchemy.orm.exc import NoResultFound

@api.errorhandler(NoResultFound)
def database_not_found_error_handler(e):
    log.warning(traceback.format_exc())
    return {'message': 'A database result was not found.'}, 404
```

INERACTIVE DEBUGGER



```python
from flask import Flask
from flask_restplus import Resource, Api

app = Flask(__name__)
api = Api(app)

@api.route('/hello')
class HelloWorld(Resource):
    def get(self):
        return {'hello': 'world'}

if __name__ == '__main__':
    app.run(debug=True)
```

Demo code and article available on my blog:

karzyn.com

THANK YOU