Princípios de Projeto REST-WS (cont.)

4. Todos os recursos acessados via HTTP **GET** não geram **efeitos colaterais**:
   - i.e., nenhuma modificação será feita no recurso acessado.
5. Representações não devem ser **"ilhas"**:
   - Representações devem conter *links* que permitam o cliente obter informações mais detalhadas ou correlatas.
6. **Revelação gradual de dados** no projeto do serviço:
   - Uso de *hiperlinks* para fornecer mais detalhes.
7. O formato das **respostas** podem ser especificados por meio de **esquemas**
   - DTD, W3C Schema, etc.

Princípios de Projeto REST-WS (cont.)

8. Descrição de como os serviços são invocados usando documento **WSDL** ou mesmo um simples **HTML**.

Prática

https://flask-restful.readthedocs.io/en/latest/

About

Flask-RESTful provides an extension to Flask for building REST APIs. Flask-RESTful was initially developed as an internal project at Twilio, built to power their public and internal APIs.

Useful Links

The Flask Website
Flask-RESTful @ PyPI
Flask-RESTful @ github
Issue Tracker

This Page

Show Source

Quick search

Go

Flask-RESTful

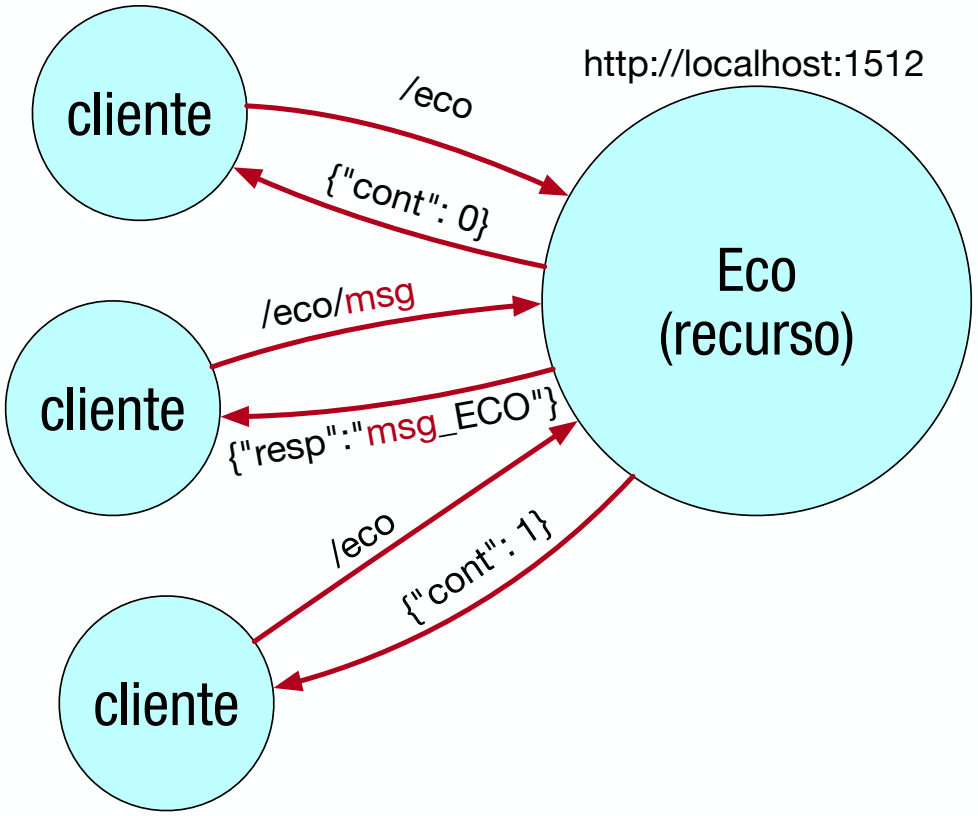
Flask-RESTful is an extension for Flask that adds support for quickly building REST APIs. It is a lightweight abstraction that works with your existing ORM/libraries. Flask-RESTful encourages best practices with minimal setup. If you are familiar with Flask, Flask-RESTful should be easy to pick up.

User's Guide

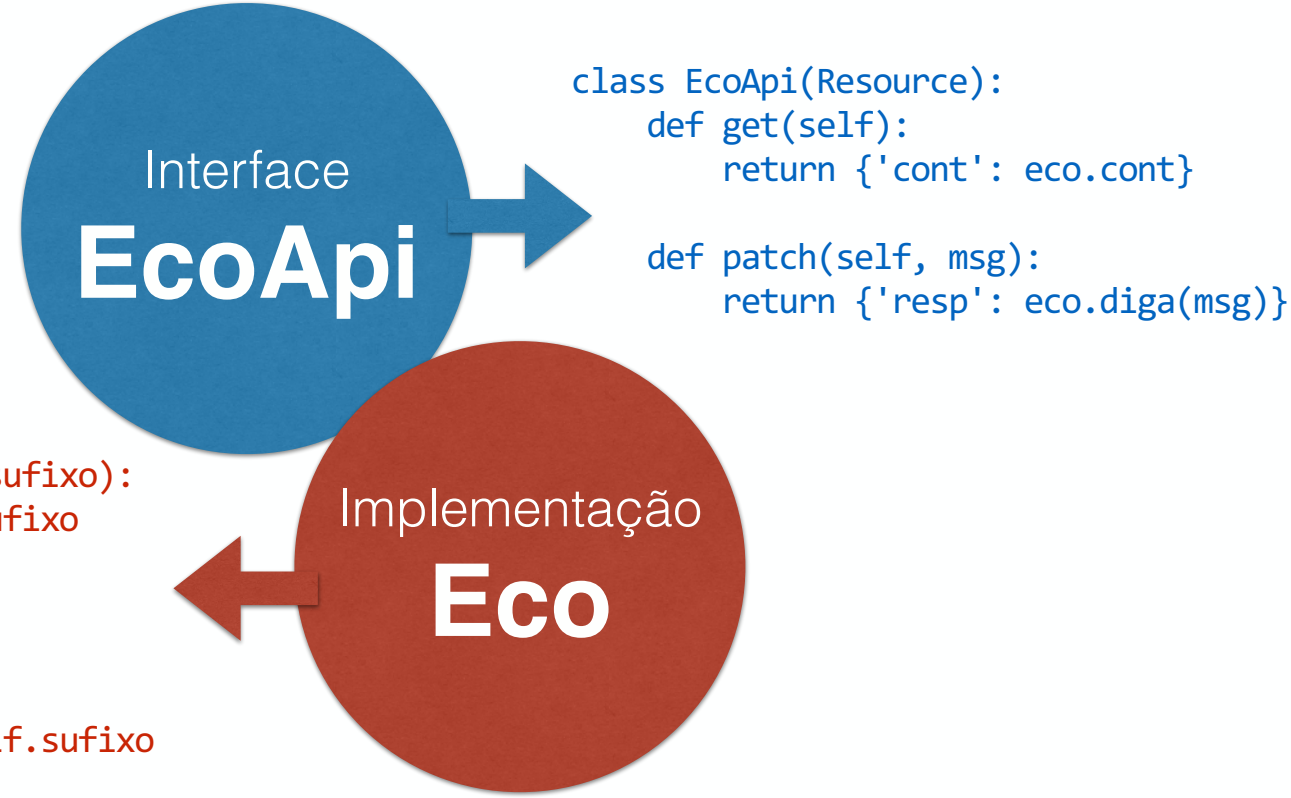
This part of the documentation will show you how to get started in using Flask-RESTful with Flask.

- Installation
- Quickstart
  - A Minimal API
  - Resourceful Routing
  - Endpoints
  - Argument Parsing
  - Data Formatting

Experimento



Separação Interface / Implementação




```python
class EcoApi(Resource):
    def get(self):
        return {'cont': eco.cont}

    def patch(self, msg):
        return {'resp': eco.diga(msg)}
```

```python
class Eco:
    def __init__(self, sufixo):
        self.sufixo = sufixo
        self.cont = 0

    def diga(self, msg):
        self.cont += 1
        return msg + self.sufixo
```

ws-rest

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

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

class Eco:
    def __init__(self, sufixo):
        self.sufixo = sufixo
        self.cont = 0

    def diga(self, msg):
        self.cont += 1
        return msg + self.sufixo

eco = Eco('_ECO')

class EcoApi(Resource):
    def get(self):
        return {'cont': eco.cont}

    def patch(self, msg):
        return {'resp': eco.diga(msg)}

api.add_resource(EcoApi, "/eco", "/eco/<string:msg>")

if __name__ == '__main__':
    app.run(port=1512, debug=True)
```

cliente curl

```
curl http://localhost:1512/eco              # GET contador
curl http://localhost:1512/eco/OLA -X PATCH   # "OLA_ECO"
```

cliente Python

```python
from requests import get, post, put, delete, patch
from sys import argv, stderr

def main():
    if len(argv) < 2:
        print(f"USO: {argv[0]} <URL> <msg>", file=stderr)
        exit(1)

    url = argv[1]
    msg = " ".join(argv[2:])

    r = patch(url + "/eco/" + msg).json()
    print("Resposta:", r['resp'])

    r = get(url + "/eco").json()
    print("Contador:", int(r['cont']))

if __name__ == '__main__':
    main()
```

Exercício - Quickstart

Table of Contents

Related Topics

This Page

Show Source

Quickstart

It's time to write your first REST API. This guide assumes you have a working understanding of Flask, and that you have already installed both Flask and Flask-RESTful. If not, then follow the steps in the Installation section.

A Minimal API

A minimal Flask-RESTful API looks like this:

```
from flask import Flask
from flask_restful import Resource, Api

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

class HelloWorld(Resource):
    def get(self):
        return {'hello': 'world'}

api.add_resource(HelloWorld, '/')

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

https://flask-restful.readthedocs.io/en/latest/quickstart.html

Prática 1: Conta Simples

- Uma vez feito o tutorial, construa um serviço de **conta bancária simples:**
  - uma única conta (com identificador, por exemplo, “1234-5”)
- API:
  - GET ➔ obtém **saldo** da conta ➔
    - ***http://localhost:XXXX/contas/1234-5***
  - PATCH ➔ **depósito** / **saque** de valores na conta ➔
    - ***http://localhost:XXXX/contas/1234-5/deposito/100.0***
- (substitua XXXX pela porta do seu serviço)

Resposta (JSON):
{ "saldo": 100.0 }

Envie o seu programa Python no AVA: **Conta RWS**.

Uso de Argumentos JSON

- Antes:
  - `curl http://localhost:XXXX/eco`
  - `curl http://localhost:XXXX/eco/ABC -X PATCH`
- Melhor:
  - `curl http://localhost:XXXX/eco -d '{ "msg":"ABC"}' -X PATCH -H "Content-Type: application/json"`
- Uso de **argumentos no corpo da mensagem** HTTP