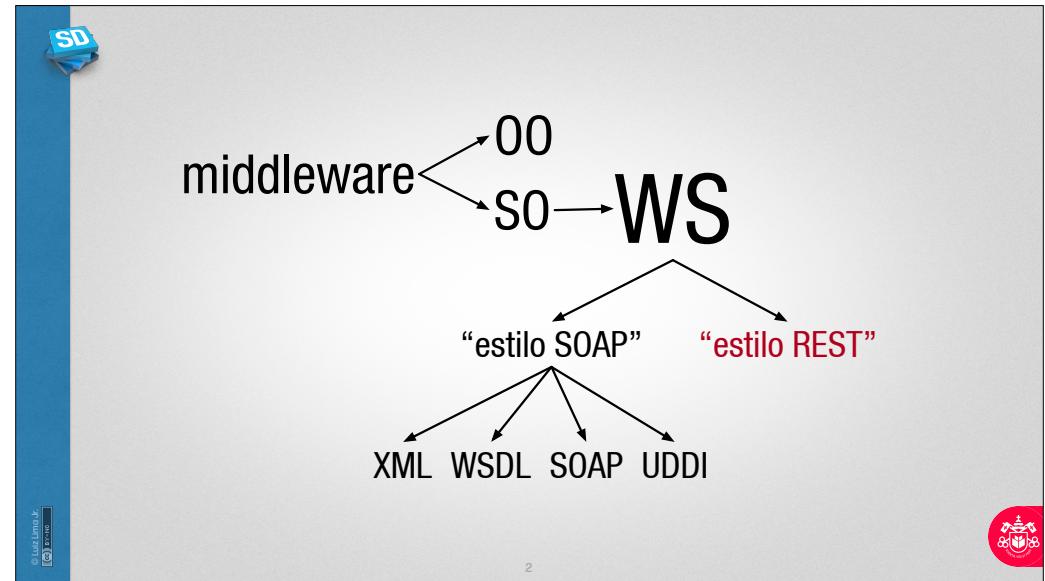


1



2

RESTful Web Services

Sistemas Distribuídos
Prof. Luiz A. de P. Lima Jr.
luiz.lima@pucpr.br
Escola Politécnica - PUCPR

3

O que é REST?

- Tese de doutorado de **Roy Fielding**
- Descreve **estilo de arquitetura** de sistemas em rede.
- REST = "**RE**presentational State Transfer"

4

Conceitos Básicos

- **Web:** formada de recursos
- **Recursos** = qualquer item de interesse (i.e., informação com *hiperlinks*)
- Recursos não são úteis a menos que se tenha uma **representação** para eles.
- Representações: **tipos MIME**
- Recursos possuem **estado**.
 - Representações devem capturar o **estado do recurso**.
- **REST:** uso de **substantivos** (“recursos”) ao invés de verbos (operações)

```
<user>
  <id>1</id>
  <name>Luiz</name>
  <profession>Teacher</profession>
</user>
```

<https://www.sitepoint.com/mime-types-complete-list/>

5



REST x SOAP

• Mais simples que SOAP

- WSDL e esquemas XML não são essenciais
- **REST:** “*Lightweight Web Services*”

~~<?xml version="1.0"?>
<soap:Envelope
 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:Body
 p1:obtemSaldo
 <p1:ID>1234-5</p1:ID>
 </p1:obtemSaldo>
 </soap:Body>
</soap:Envelope>~~

REST

<http://www.banco.com/contas/1234-5>

SOAP

6



REST x SOAP (cont.)

- **Equivalentes em funcionalidade:**
 - Tudo que pode ser feito com SOAP, pode também ser feito com uma arquitetura RESTful.
- **SOAP** = “**carta**” (c/ envelope)
- **REST** = “**cartão postal**”
(gasta menos “papel”, i.e., largura de banda)
- Mesmo nível de **segurança** que SOAP

7



REST x SOAP (cont.)

- Não usa **XML** necessariamente (especialmente para requisições):
 - Formatos possíveis:
 - CSV (Comma-Separated Values)
 - JSON (JavaScript Object Notation)
- Recursos x métodos ou serviços:
 - **SOAP:** `obtemProduto()` + `obtemPreço()`
 - **REST:** `produto` = recurso com todas as informações (ou *links* para elas)

8



Exemplo de Recurso

- Livro de SD em uma biblioteca
- Acesso ao **recurso** (lista):
 - <http://www.bib.com/sd>
- Exemplo de **representação** do recurso:
 - `sd.html` (MIME: `text/html`)
- A representação coloca o cliente em um **ESTADO**

(obtida no ponto
de acesso)

9



Exemplo de Recurso (cont.)

- Ao acessar um recurso e obter a sua representação, o **cliente muda de estado**.
- REST, segundo Roy Fielding:
 “REST procura capturar a imagem de como **aplicações web bem projetadas** se comportam: uma rede de **páginas web** (uma máquina virtual de estados), onde o usuário progride em uma aplicação por meio da **seleção de links** (transições de estado) resultando na **próxima página** (representando o próximo estado da aplicação) sendo transferida ao usuário e apresentada para o seu uso.”

10



Motivação

- Capturar as **características da web** que a fizeram bem sucedida.
- Usar tais características para conduzir a evolução da web: **WS**

11



REST

- REST não é um **padrão**
 ○ Não há especificação como SOAP, XML-RPC, etc.
- É um “**estilo arquitetural**”
 ○ assim como o modelo “cliente/servidor”, que não é definido como um “padrão”
- Embora não seja um padrão, **usa padrões**:
 - HTTP
 - URL
 - XML/JSON/HTML/PNG/JPEG (representações de recursos)
 - text/xml, text/html, image/jpg, etc. (tipos dos recursos – MIME types)
- REST **oculta detalhes de implementação** (Java? CGI?) do serviços web.

12



SD

HTTP e REST - Exemplo

Método HTTP	Ação	Exemplo
GET	Obter informação sobre recursos	http://banco.com/contas (obtém lista de contas)
GET	Obter informação a respeito de um recurso	http://banco.com/contas/1234-56 (obtém saldo conta "1234-56")
POST	Cria um novo recurso	http://banco.com/contas (cria nova conta a partir de dados fornecidos com a requisição)
PUT	Modifica um recurso	http://banco.com/contas/1234-56/deposito?valor=100.00 (efetua depósito na conta)
DELETE	Deleta um recurso	http://banco.com/contas/1234-56 (remove conta "1234-56")

13

13



14

SD

Exemplo de Serviço Web Estilo REST

- Biblioteca (fictícia)
- Serviço web para:
 - **obter uma lista de livros** sobre Sistemas Distribuídos
 - **obter informações detalhadas** sobre um livro particular
 - submeter um **pedido de reserva** do livro
- Como tais serviços poderiam ser implementados no estilo “RESTful”?

15

15

SD

Exemplo REST - Obtendo a Lista de Livros

- O serviço web torna disponível a URL do recurso da lista de livros:
 - <http://www.bib.com/sd>
- Observações:
 - A forma **como a lista de livros é gerada** é algo **transparente** para o cliente.
 - Sendo transparente para o cliente, a implementação subjacente deste recurso **pode ser alterada** sem afetar os clientes (que sabem apenas que se submeterem a URL obterão a lista de livros):
 - “fraco acoplamento”

16

Exemplo REST - Obtendo a Lista de Livros (cont.)

- O que o cliente recebe (supondo resposta em XML):

```
<?xml version="1.0"?>
<p:Livros xmlns:p=http://www.bib.com
  xmlns:xlink=http://www.w3.org/1999/xlink
  <Livro id="00123" xlink:href=http://www.bib.com/sd/00123/>
  <Livro id="00124" xlink:href=http://www.bib.com/sd/00124/>
  <Livro id="00125" xlink:href=http://www.bib.com/sd/00125/>
</p:Livros>
```

17



Exemplo REST - Obtendo a Lista de Livros (cont.)

- O que o cliente recebe (supondo resposta em JSON):

```
{
  "livros": [
    {"id": "00123", "xlink": href=http://www.bib.com/sd/00123/},
    {"id": "00124", "xlink": href=http://www.bib.com/sd/00124/},
    {"id": "00125", "xlink": href=http://www.bib.com/sd/00125/}
  ]
}
```

18



Exemplo REST - Obtendo a Lista de Livros (cont.)

- Observações:

A lista contém as URLs de onde obter detalhes sobre cada livro.

- Chave para REST:

O cliente muda de um estado para o próximo baseado ***na análise e escolha das alternativas fornecidas*** na resposta que obteve anteriormente.

19



Exemplo REST - Detalhes de um livro

- Para solicitar detalhes (e.g., índice) do livro 00124:

<http://www.bib.com/sd/00124>

- Resposta possível (XML):

```
<?xml version="1.0"?>
<p:Livro xmlns:p=http://www.bib.com xmlns:xlink=http://
www.w3.org/1999/xlink
  <Livro-ID>00124</Livro-ID>
  <Titulo>SD</Titulo>
  <Descricao>Livro sobre Sistemas Distribuídos</
Descricao>
  <Indice xlink:href="xlink:href=http://www.bib.com/
sd/00124/indice"/>
  <Disponibilidade>5</Disponibilidade>
</p:Livro>
```

20



19

20

Exemplo REST - Detalhes de um livro (cont.)

- Observações:
 - Resposta possui [link para mais detalhes](#) ([Índice](#)).
 - Usuário pode se aprofundar ainda mais nos detalhes do livro.

21



Exemplo REST - Reserva de um livro

- O serviço web da Biblioteca disponibiliza uma **URL para solicitação de reserva** de livro.
- O cliente **cria um documento** de solicitação que esteja de acordo com o esquema que a Biblioteca definiu (e.g., **WSDL** ou **Descrição textual em HTML**).
- O cliente **envia a requisição** (e.g., em XML) como um **POST HTTP**.
- O serviço responde à requisição com **uma URL** para a sua solicitação:
 - O cliente desta forma pode **consultar/atualizar** a sua requisição.
- **A requisição gera um novo serviço web** oferecido pelo servidor em um novo endereço (URL).

22



Características de Serviços Web Estilo REST

23

Características de Serviços Web Estilo REST

- URLs não devem idealmente revelar a **técnica de implementação** usada para prover o serviço web:
 - <http://www.bib.com/sd/00124> **ao invés de** <http://www.bib.com/sd/00124.xml>
- Características:
 - Cliente/servidor:
 - **modelo de interação pull** (componentes consumidores “puxam” as representações)
 - Sem estado (**stateless servers**):
 - Cada requisição do cliente deve conter toda informação necessária para a compreensão da requisição, não podendo fazer uso de nenhuma informação eventualmente salva no servidor.

24



Características de Serviços Web Estilo REST (cont.)

Cache:

- Para otimizar a eficiência da rede, as respostas devem poder ser etiquetadas “**cacheable**” ou “**non-cacheable**”.

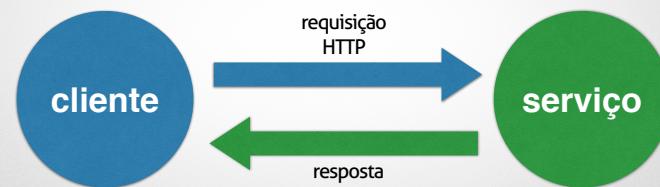
Interface Uniforme:

- Todos os recursos são acessados por meio de interfaces genéricas – e.g. HTTP **GET** (lê um recurso), **POST** (cria um recurso), **PUT** (substitui um recurso existente), **PATCH** (atualiza um recurso existente) **DELETE** (remove um recurso).

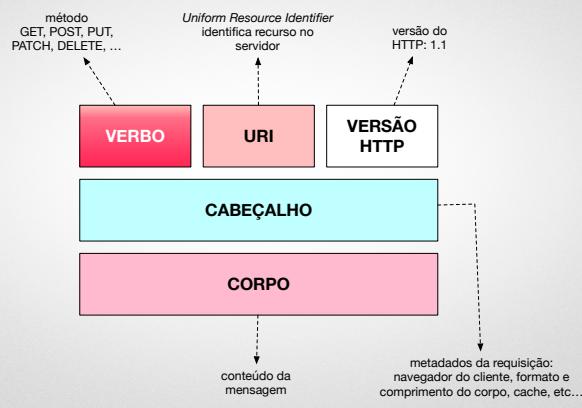


Mensagens e HTTP

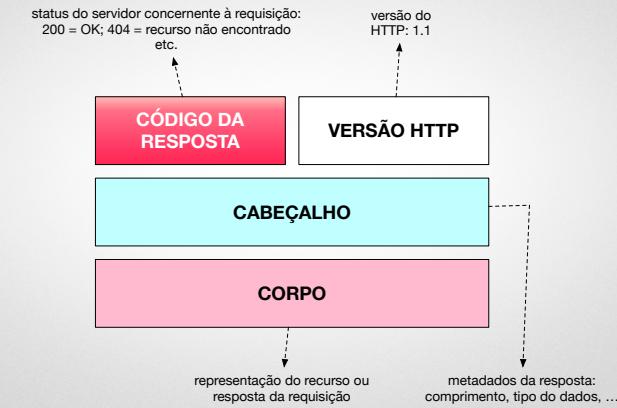
- REST utiliza o protocolo HTTP como meio de comunicação entre clientes e serviços.



Uma REQUISIÇÃO HTTP



Uma RESPOSTA HTTP





HTTP e REST - Exemplo

Método HTTP	Ação	Exemplo
GET	Obter informação sobre recursos	http://banco.com/contas (obtém lista de contas)
GET	Obter informação a respeito de um recurso	http://banco.com/contas/1234-56 (obtém saldo conta "1234-56")
POST	Cria um novo recurso	http://banco.com/contas (cria nova conta a partir de dados fornecidos com a requisição)
PUT PATCH	Modifica um recurso	http://banco.com/contas/1234-56/deposito?valor=100.00 (efetua depósito na conta)
DELETE	Deleta um recurso	http://banco.com/contas/1234-56 (remove conta "1234-56")

29



Características de Serviços Web Estilo REST (cont.)

- Recursos são **nomeados**:
○ O sistema é composto de recursos que possuem nomes definidos (referenciados por URLs).
- Representações **interconectadas** de recursos:
○ As **representações** de recursos são **interconectadas usando URLs**, permitindo assim clientes passarem de um estado ao próximo.
- Componentes em **níveis**:
○ Intermediários tais como **proxies**, **caches**, **gateways**, etc. podem ser inseridos entre clientes e recursos de forma a melhorar **desempenho**, **segurança**, etc.

30



Princípios de Projeto de Serviços Web Estilo REST

31



Princípios de Projeto REST-WS

1. **Chave** para criação de serviços web no estilo REST: **identificação de todas as entidades conceituais** ("recursos") que precisam ser expostas como serviços.
○ e.g. Lista de livros, detalhes do livro, requisição de reserva, contas, ...
2. Criação de uma **URL** para cada recurso:
○ Recursos são substantivos, não verbos
3. Categorização dos recursos:
○ Usuários poderão:
 - Receber representações dos recursos (HTTP GET); ou
 - Adicionar novos recursos, modificá-los, etc. (HTTP POST, PUT, PATCH e/ou DELETE)

32



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. Nenhuma representação deve ser uma **ilha**:
◎ 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 *hyperlinks* para fornecer mais detalhes.
7. O formato das **respostas** podem ser especificados por meio de **esquemas**
◎ DTD, W3C Schema, etc.

33



Princípios de Projeto REST-WS (cont.)

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

34



Prática

35

<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



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

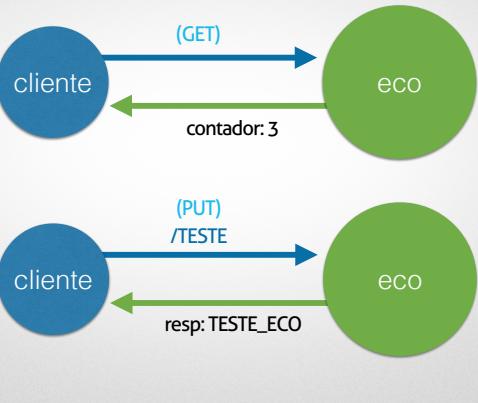
This part of the documentation will show you how to get started in using Flask-RESTful with Flask.

- [Installation](#)
- [Quickstart](#)
 - [A Minimal API](#)
 - [Resourceful Routing](#)
 - [Endpoints](#)
 - [Argument Parsing](#)
 - [Data Formatting](#)



36

Experimento



37

```
SD
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 put(self, msg):
        return {'resp': eco.diga(msg)}

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

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

WS-rest

cliente curl

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

cliente Python

```
from requests import get, post, put, delete
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 = put(url + "/eco/" + msg).json()
    print("Resposta:", r['resp'])

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

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

38

Exercício - Quickstart

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>

Table of Contents

- A Minimal API
- Resourceful Routing
- Endpoints
- Argument Parsing
- Data Formatting
- Full Example

Related Topics

- Documentation overview
- Previous: Installation
 - Next: Request Parsing

This Page

Show Source

39

Prática

API:

◎ GET → saldo

◎ PATCH → depósito / saque

URL:

◎ <http://localhost:PORTA/ID>

Detalhes: roteiro de lab. (página web)

40

Referências

- ◎ Roger L. Costello, “**Building Web Services the REST Way**”, xFront.
- ◎ WikiPedia, “**Representational State Transfer**”, https://en.wikipedia.org/wiki/Representational_state_transfer
- ◎ M. Kalin, “**Java Web Services: Up and Running**”, 1st edition, O'Reilly, Feb. 2009.
- ◎ M. Elkstein, “**Learn REST: A Tutorial**”, <http://rest.elkstein.org/2008/02/what-is-rest.html>
- ◎ M. Grimberg, ”**Designing a RESTful API with Python and Flask**”, <http://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask>



41