Desmistificando as aplicações RESTful usando o Django Rest

Bruno Oliveira | Instituto Federal da Bahia

Bruno Oliveira

oliveirabrunoa@gmail.com

Bruno.Oliveira21

oliveirabrunoa

http://lattes.cnpq.br/9651894815550789

O que iremos aprender?

Agenda

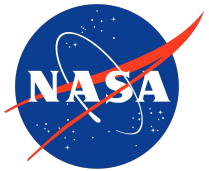
- Django Framework
  - Overview (Breve histórico, estrutura, caracteristicas etc…)
  - Exemplo Básico - primeiros passos
- Entendendo o Estilo Arquitetural REST
  - Principios e Terminologias REST
  - REST x SOAP
  - Verbos HTTP
  - Vantagens x Desvantagens
  - Quando Utilizar e Quando Não Utilizar
- Django REST
  - Exemplo Prático (Clone github)

Um pouco de história...

- O Django foi criado originalmente para gerenciar publicações jornalísticas do *Word Online* para a construção de sites interativos em curtos períodos de tempo. (2003)
- ...Os desenvolvedores começaram a extrair um framework de desenvolvimento web genérico capaz de construir aplicações Web cada vez mais rápido.
- Em 2005, após constantes melhorias no framework, o jornal decidiu abrir o código-fonte do software resultante, o Django.

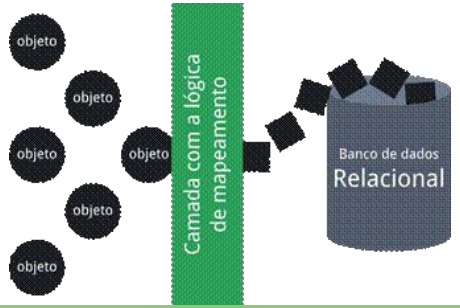
Guitarrista de jazz Django Reinhardt

Quem usa o Django?



Entre outros...https://www.shuup.com/en/blog/25-of-the-most-popular-python-and-django-websites/

Por que usar o Django?

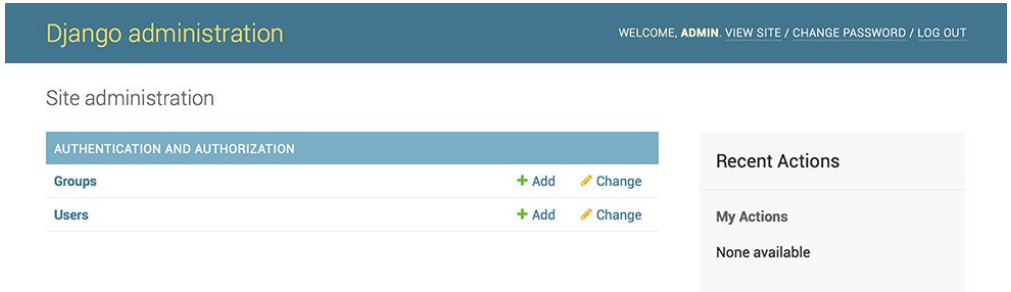
- Framework web ágil
  - “Nós fazemos a parte repetitiva, você faz o que interessa.
- Utiliza o princípio *Don’t repeat yourself* (DRY)
  - Apps plugáveis;
- Mapeamento objeto-relacional;
  - 

Por que usar o Django?

- Urls elegantes;
  - Url sem padrão inteligível:

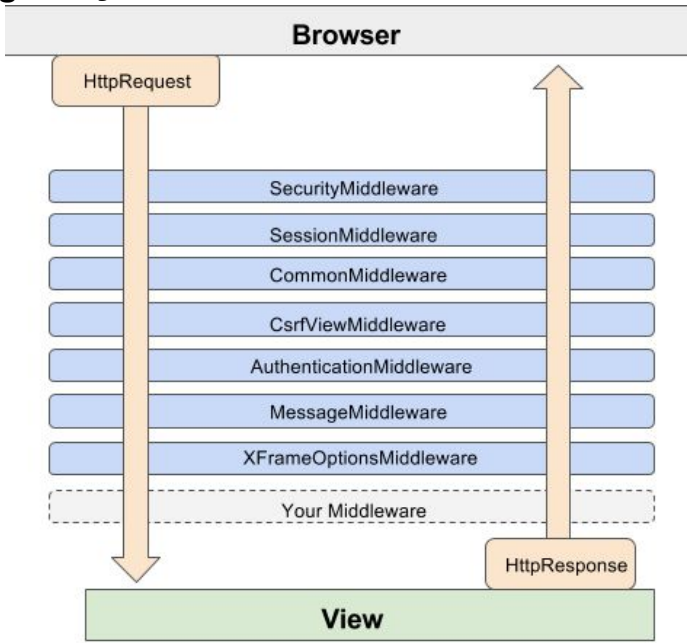
    “ME2/Sites/dirmod.asp?sid=&type=gen&mod=Core+Pages&gid=A6CD4967199A42D9B65B1B”

    No Django: `/newsarchive/<year>/<month>/`.

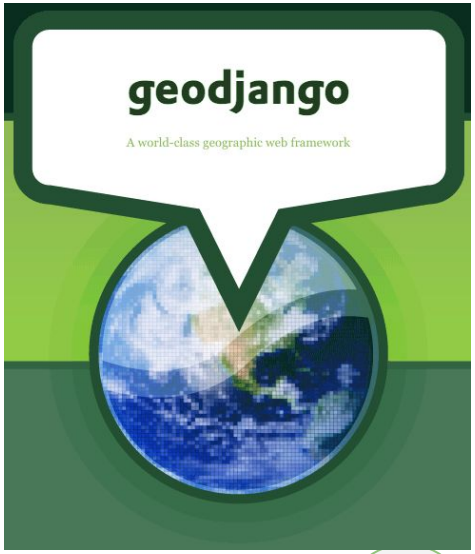
- Interface administrativa automática que inclui sistema de autenticação e gerenciamento de permissões;
  - 

E muito mais...

- Conjunto de middlewares voltados para segurança



- Geodjango



E muito mais...

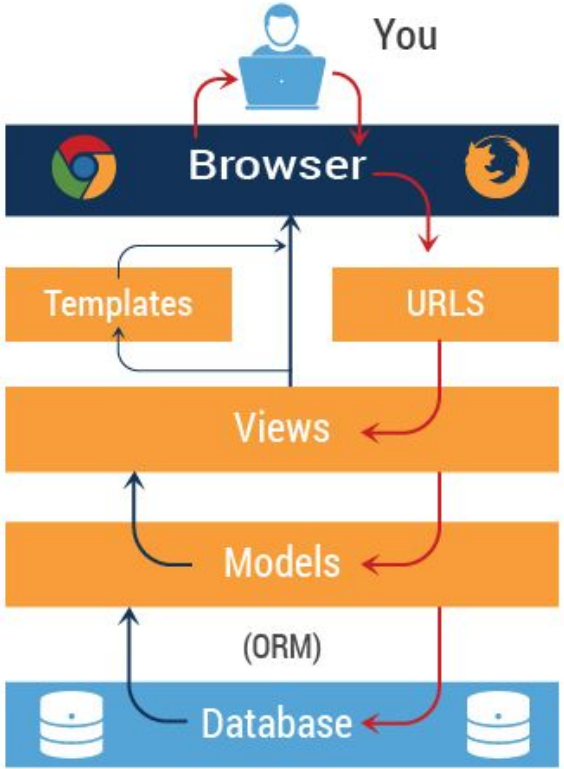
- Geração automática de formulários;
- Sistema de templates flexível;
- Sistema de cache;
- Geodjango framework;
- Internacionalização
- Comunidade ativa.

MVC vs MTV

“Se você está faminto por acrônimos, você deve dizer que Django é um framework MTV”

“No final do dia, é claro, tudo se resume a conseguir fazer coisas. Independentemente como as coisas são nomeadas, Django as executa da forma que é mais lógica para nó

Ciclo de requisição



Exemplo Prático

Configurando o ambiente

- **Criando e ativando um ambiente virtual:**
  - Criar diretório: mkdir <<nomedapasta>>
  - Criar um ambiente virtual dentro da pasta: *python3 -m venv <<myvenv>>*
  - Ativar o ambiente: *source myvenv/bin/activate*
- **Instalando o Django Framework:**
  - pip install django

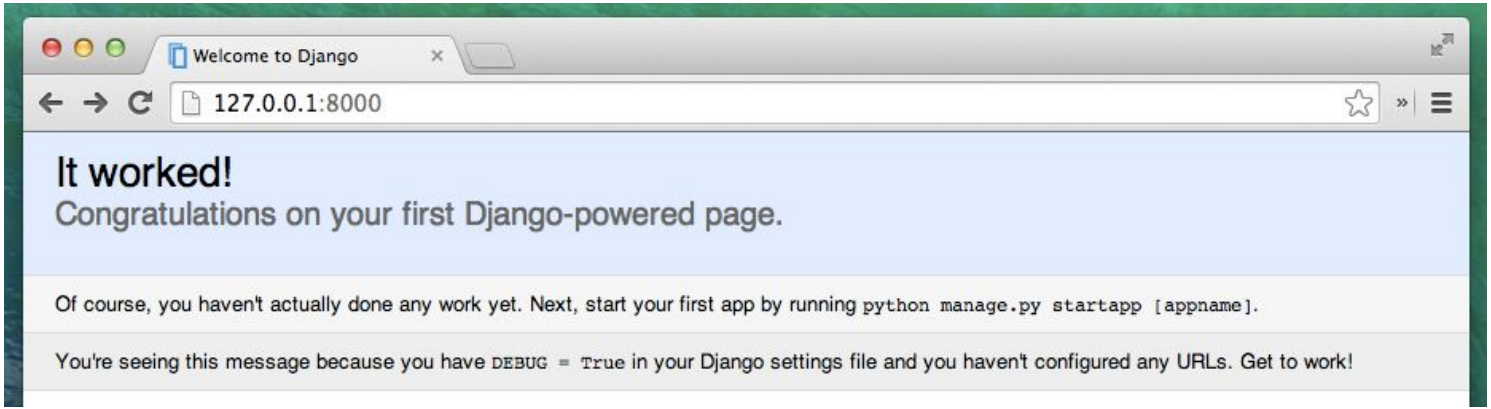
Criando projeto e aplicação

- **Criando projeto:**
  - django-admin startproject <<nomeprojeto>>
- **Criando app:**
  - Acesse o diretório que tem o <<nomedoprojeto>>
  - Crie um app: python manage.py startapp <<app>>
  - Execute o comando *migrate* para gerar a interface administrativa automaticamente:

    python manage.py migrate

Criando projeto e aplicação

- **Crie um super usuário e inicie o servidor:**
  - python manage.py createsuperuser
  - python manage.py runserver
- **Acesse 127.0.0.1:8000.**



Criando projeto e aplicação

- **Interface administrativa**

Criando projeto e aplicação

- **Instale o app criado em settings.py**

settings.py ×

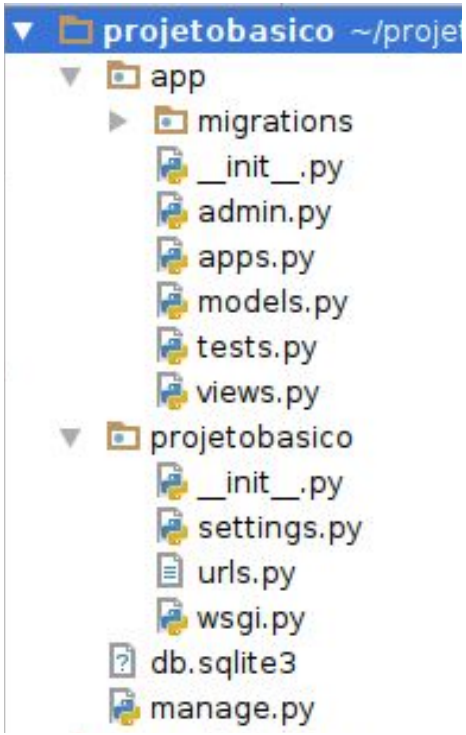
```
# SECURITY WARNING: don't run with debug
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app',
]
```

Estrutura do projeto Django



Modelos

- Um modelo define a estrutura dos objetos da aplicação. Ele contém os campos e comportamentos essenciais dos dados que você está armazenando;
- Cada modelo é uma classe Python que herda django.db.models.Model;
- Cada atributo de um modelo representa um campo no banco de dados.

Exemplo:

```python
from django.db import models

class Livro(models.Model):
    id_livro = models.AutoField(primary_key=True, blank=False, null=False)
    titulo = models.CharField(max_length=250, verbose_name="Titulo",blank=False, null=False)
    autor = models.CharField(max_length=150, verbose_name="Autor",blank=False, null=False)
    editora = models.CharField(max_length=150, verbose_name="Editora",blank=True, null=True)
    ano_publicacao = models.IntegerField(verbose_name="Ano",blank=True, null=True)
    edicao = models.IntegerField(verbose_name="Edição",blank=False, null=False)
    isbn = models.CharField(max_length=13, verbose_name="ISBN",blank=False, null=False)
    sinopse = models.TextField(verbose_name="Sinopse",blank=True, null=True)
```

View

- Uma função do tipo view é uma função em python que recebe uma requisição (*request*) e retorna uma resposta (*response*).
- Essa resposta pode ser o conteúdo HTML de uma página web, um redirecionamento, uma imagem...
- A própria função view contém toda a lógica necessária para consultar os dados através do model e retornar uma resposta.
- Por convenção, essas funções do tipo view ficam em um arquivo chamado **views.py**

Exemplo:

```
from django.shortcuts import render, render_to_response
from .models import Livro

# Create your views here.

def listar_livros(request):
    livros = Livro.objects.all()
    return render_to_response("livros.html", {"livros":livros})
```

Urls

- Um esquema de URL elegante e limpo, é um detalhes importante em uma aplicação Web!
- No Django, nós usamos algo chamado URLconf (configuração de URL), que é um conjunto de padrões que Django vai tentar coincidir com a URL recebida para encontrar a visão correta. (Expressões regulares simples)

Exemplo:

```
from django.conf.urls import url, include
from . import views

urlpatterns = [

    url(r'^listar_livros/$', views.listar_livros),
]
```

QuerySets

- Uma vez que tenha criado seu modelos de dados, o Django automaticamente fornece uma API de abstração do banco de dados que permite a criação,consulta, edição e deleção objetos.

Exemplos:

```
>>> from administracao.models import Livro
>>> livro = Livro(titulo='Sistemas Operacionais Modernos', autor='Andrew S. Tanenbaum',
edicao=2, ....)
>>> livro.save() #Cria novo registro

>>> livros = Livro.objects.all() #Busca todos os objetos Livro no banco de dados
>>> first_livro = Livro.objects.get(pk=1) #Retorna o primeiro livro cadastrado

>>> first_livro = Livro.objects.get(pk=1)
>>> first_livro.delete() #deleta o livro de pk 1
```

Formulários

- No Django, os formulários podem ser criados “do zero”, isto é, a partir da especificação de cada tipo de dado para um modelo
- Ou, usando modelform (recomendado)

Exemplo:

```
from django import forms
from .models import Livro


class LivroForm(forms.ModelForm):

    class Meta:
        model = Livro
        #fields =('titulo', 'autor', 'isbn') Declaração dos fields explicitamente,ou..
        fields = '__all__'
```

Templates

- A linguagem de templates do django é formada por tags e filtros;
- O conceito de herança de templates permite reaproveitar trechos de outros templates seguindo a mesma lógica da herança em POO; `{% extends admin/base.html' %}`
- As tags (com comandos) devem estar entre as marcações {% %} e os dados {{ dados }}

Exemplo:

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Exemplo Template Django</title>
</head>
<body>
    {% csrf_token %}

    {% for livro in livros %}

    <h1> {{ livro.titulo }}</h1>
    {% endfor %}

</body>
</html>
```

Não acaba por aqui...

Django - Documentação Oficial:
https://docs.djangoproject.com/en/1.10/

Django Girls - Tutorial em Português:
https://tutorial.djangogirls.org/pt/

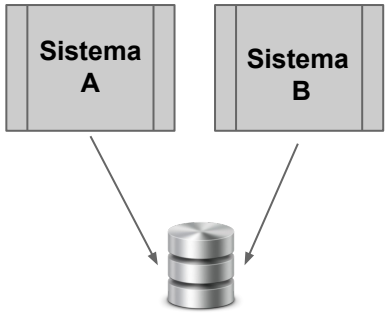
O Estilo Arquitetural REST

Por Bruno Oliveira

O problema...

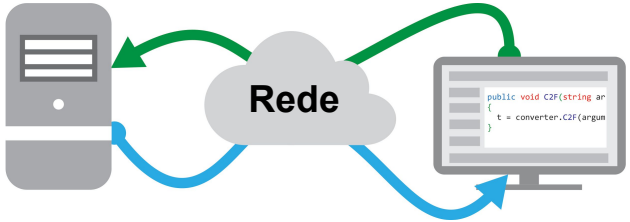
Como integrar sistemas de tecnologias e/ou plataformas diferentes?

Solução Prática 1: Banco de Dados Compartilhado



Solução Prática 2: Transferência de Arquivos

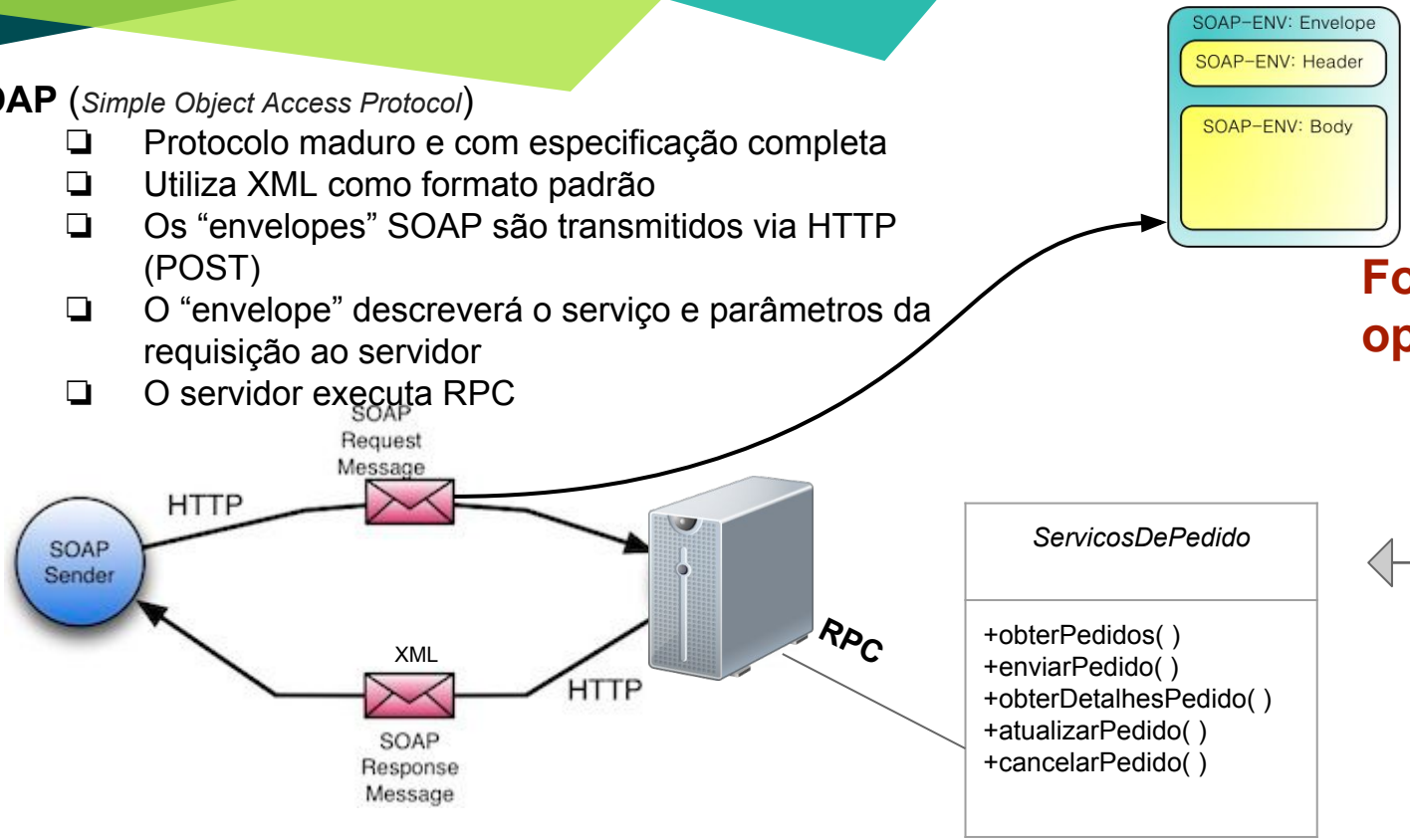
Solução Prática 3: Web Service



Um pouco de história...

SOAP (*Simple Object Access Protocol*)

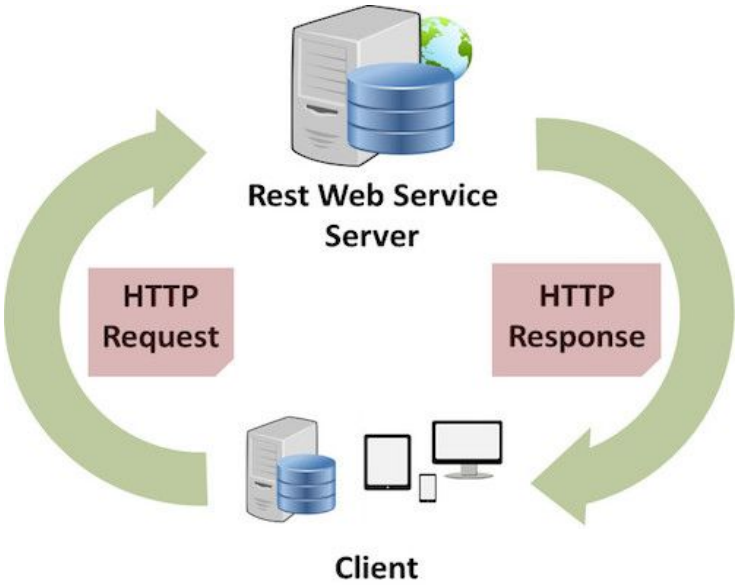
- Protocolo maduro e com especificação completa
- Utiliza XML como formato padrão
- Os “envelopes” SOAP são transmitidos via HTTP (POST)
- O “envelope” descreverá o serviço e parâmetros da requisição ao servidor
- O servidor executa RPC



O Estilo Arquitetural REST

REST (*Representational State Transfer*)

- "Um estilo arquitetural para sistemas de hipermídia distribuídos"
  - Ex: a própria WEB!



Mas... como o servidor vai saber qual serviço deve realizar?

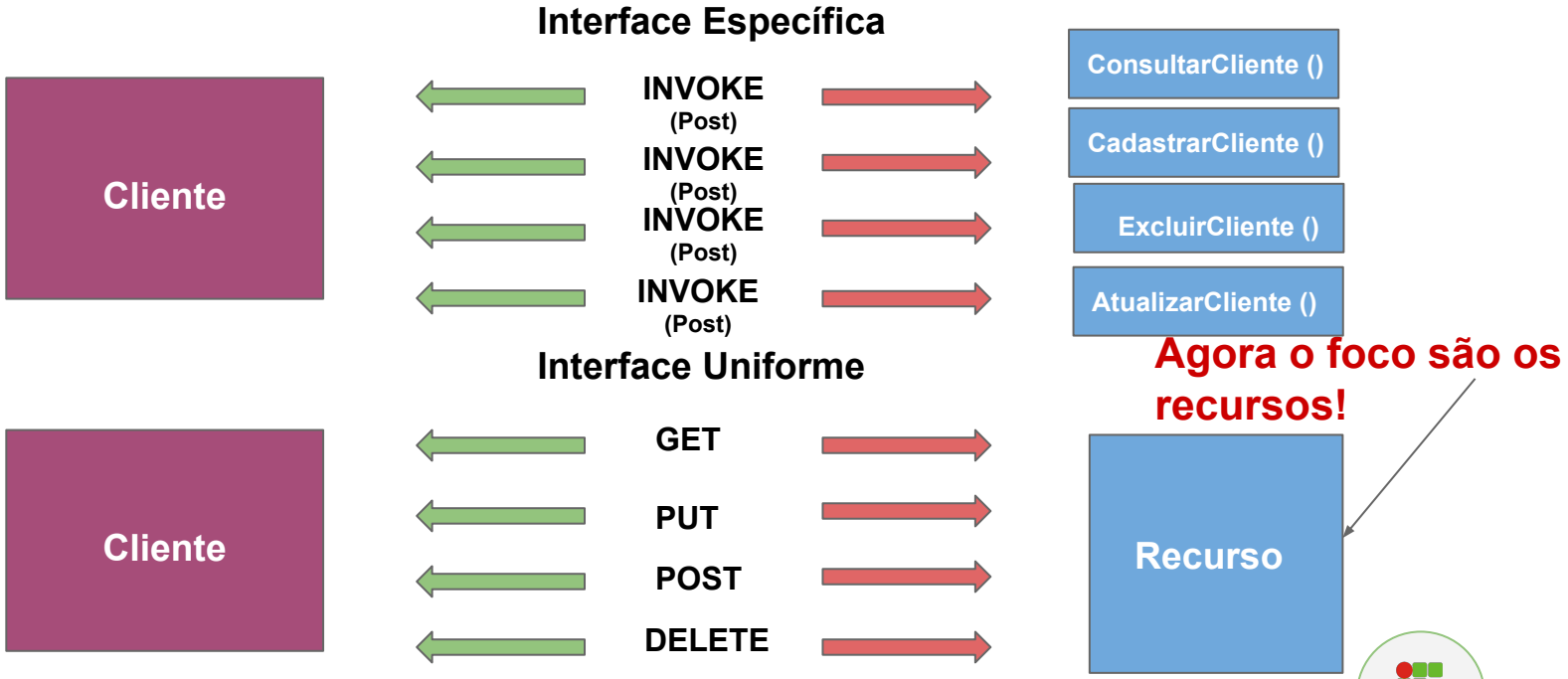
Através dos métodos do protocolo HTTP!

Métodos HTTP

- **Verbos HTTP - Operações:**
  - **GET** – recupera a representação de um recurso.
  - **POST** – adiciona informações usando o recurso da URI passada. Pode adicionar informações a um recurso existente ou criar um novo.
  - **PUT** – Atualiza um recurso na URI especificada.
  - **DELETE** – remove o recurso representado pela URI passada.

A principal diferenteça entre POST e PUT é que o primeiro pode lidar não somente com recu
mas pode fazer processamento de informações, por exemplo.

O que muda na prática?



E se algo der errado?

- Código de Status HTTP:
  - 100 - Continue
  - 200 - OK
  - 201 - Created
  - 204 - No Content
  - 304 - Not Modified
  - 400 - Bad Request
  - 401 - Unauthorized
  - 404 - Not Found
  - 405 - Method Not Allowed
  - 500 - Internal Server Error
  - ...e muitos outros https://httpstatuses.com/

Qual das abordagens é a melhor?

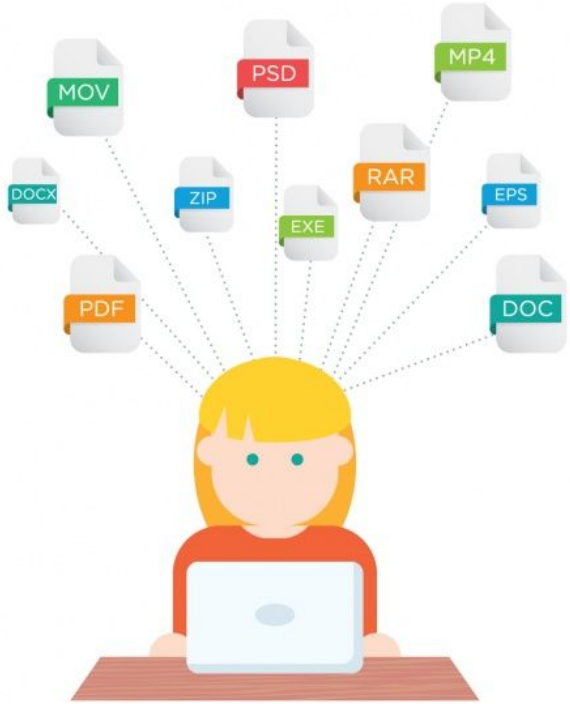
Depende...

Ambos possuem vantagens e desvantagens!

A melhor abordagem a ser utilizada depende dos requisitos do software.

Principios e Terminologias do REST

- **Recursos**
  - Qualquer coisa considerada relevante para sua aplicação
    - Ex: Relatórios, Fotos, Vídeos, Recibos, Produtos, Automóveis, Lista de Buracos nas ruas de Salvador….
    - **TUDO** é um **RECURSO**
- **“Um recurso é um objeto ou serviço que pode ser identificado através de uma URI”**



Principios e Terminologias do REST

- **Identificação do Recurso***(Addressability)*:
  - Os recursos serão identificados através de uma URL que permitirá a localização e realização de operações sobre ele.

- **Interface Uniforme**
  - As operações são baseadas no protocolo HTTP (GET, POST, PUT, DELETE)

Principios e Terminologias do REST

- **Comunicação sem estado** *(Stateless)*
  - **O servidor não manterá histórico de requisições!**
  - Dados relativos ao cliente (estado da aplicação) devem ser mantidos no próprio cliente
  - Dados relativos ao recurso devem ser mantidos no servidor.

- **Cache** *(Cacheable)*
  - O uso de Cache no cliente auxilia no economia de processamento a partir do armazenamento dos resultados das requisições
    - Obs: dada a sua complexidade, este tópico não será abordado nesse minicurso.

Principios e Terminologias do REST

- **Representações**
  - “Os recursos são dissociados de suas representações para que seu conteúdo possa ser acessado em uma variedade de formatos…”
  - Ao receber uma requisição, o servidor deverá enviar uma resposta contendo um documento (que pode ser uma página HTML, arquivo TXT, XML, JSON, XHTML…).

Estilo Arquitetural REST

Principais vantagens :

- Protocolo de transferência maduro e amplamente utilizado (HTTP)
- Escalabilidade e Alta Performance
- Uso da infraestrutura da Web a favor da aplicação
- “Facil” Implementação
- Flexibilidade de representação de recursos em diferentes formatos
- Pode ser adotado por praticamente qualquer Cliente que suporte HTTP/HTTPS (Interoperabilidade)

Mas...seria o REST a solução de todos os problemas?

Não!

Embora o REST proporcione vantagens bastante atrativas, a escolha de utilizá-lo deverá levar em conta os requisitos do software!

Quando utilizar e quando não utilizar

Quando Utilizar:

- **Além de todos os casos anteriormente citados:**
  - **Quando há limitação de recursos ou de largura de banda;**
  - **Quando a natureza da aplicação permite o uso de cache;**
  - **Operações não precisam ser continuadas (stateless);**

Quando Não Utilizar:

- **Quando o sistema realiza processamento de chamada assíncronos;**
- **For necessário utilizar outro protocolo de transporte;**
- **Quando há necessidade de estabelecer contratos formais entre o cliente e o servidor.**
- **E, obviamente, quando o serviço for stateful.**

Django REST Framework (DRF)

Por Bruno Oliveira

O que é o DRF?

- É um conjunto de ferramentas construídas a partir do Django, que facilitam a construção de API's REST.
- Essa apresentação terá foco no funcionamento básico do serviço REST do framework. Entretanto, existem váaaaarias configurações avançadas que devem ser consultadas através da documentação do Django REST Framework (http://www.django-rest-framework.org/)

Instalação

- Instale o pacote DRF:
  - pip install djangorestframework
- Registre a aplicação:

```
INSTALLED_APPS = (
    ...
    'rest_framework',
)
```

Serializers

- A serialização de objetos tem como finalidade principal salvar o estado de um objeto para ser capaz de recriá-lo quando necessário.
- Existem diversas formas de serializar objetos no Django Rest Framework. Neste tutorial, usaremos apenas JSON.
- ModelSerializers utiliza um model como meta class para serializar objetos.

Exemplo:

```
class LivroSerializer(serializers.ModelSerializer):

    class Meta:
        model = Livro
        fields = '__all__'
```

Requisições e Respostas

- **APIVIEW:**
  - As funções do tipo view, definidas em views.py, herdam da classe genérica APIVIEW.
  - Os métodos da classe tratam os principais verbos do protocolo HTTP (POST,GET,PUT,DELETE)

Exemplo:

```
class Livros(APIView):

    def get(self, request):

        livros = Livro.objects.all()
        id_livro = self.request.query_params.get('pk', None)

        if id_livro:
            livros = Livro.objects.filter(id_livro=id_livro)

        serializer = LivroSerializer(livros, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)
```

URL

- **Agora que ja temos uma APIVIEW criada, devemos definir uma URL associada a ela:**

Exemplo:

```
urlpatterns = [

    url(r'^livros/$', views.Livros.as_view()),
    url(r'^clientes/$', views.Clientes.as_view()),

    url(r'^reservas/(?P<pk>[0-9]+)/$', views.Reservas.as_view()),

    url(r'^livro_details/(?P<pk>[0-9]+)/$', views.LivroDetails.as_view()),

    url(r'^cliente_details/(?P<pk>[0-9]+)/$', views.ClienteDetails.as_view(), name='cliente_detail'),
```

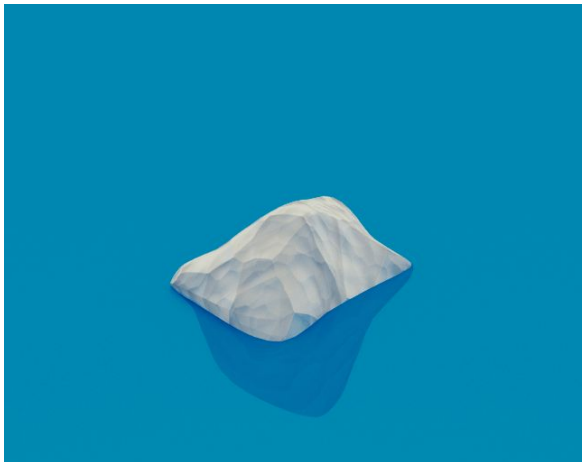
Interface REST (Browser)

Essa é só a ponta do Iceberg

O Django REST Framework possui configurações mais avançadas que não foram abordadas nesse minicurso

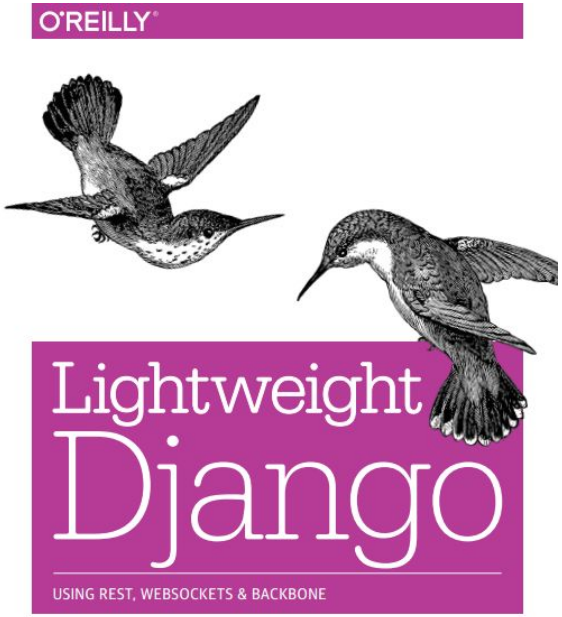
Dica de estudo:

- Mecanismos de autenticação e permissão
- Estudo de outras Views (Aqui usamos apenas APIVIEW)
- Utilizar outras representações além do JSON
- Relacionamentos e rotas

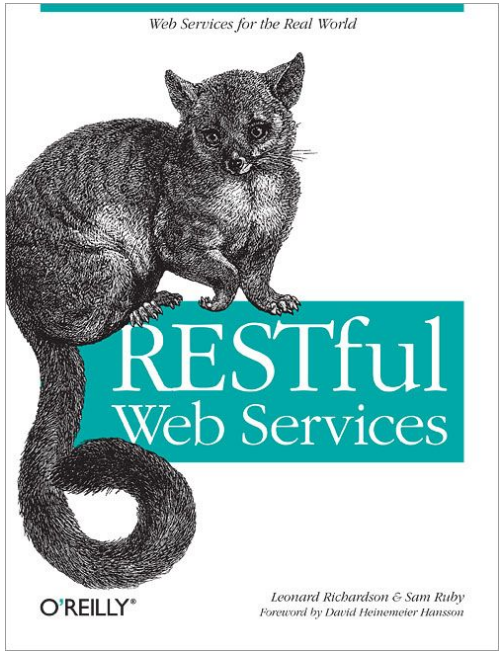


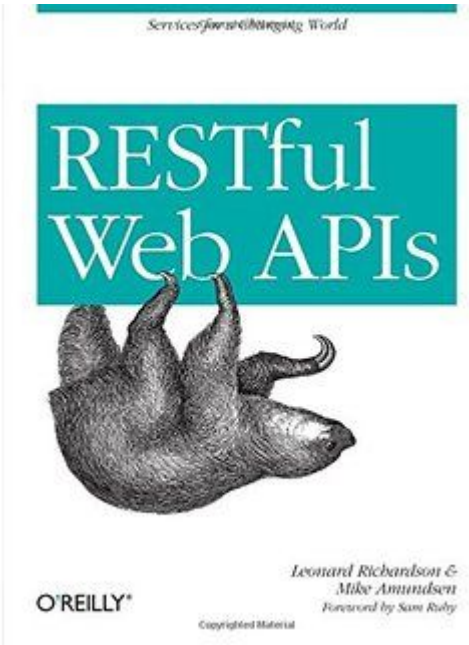
Dúvidas?

Referências



Julia Elman & Mark Lavin





Referências

https://docs.djangoproject.com/en/1.10/
http://www.slideshare.net/fernandogrd/arquivo-27975959
http://pt.slideshare.net/osantana/curso-de-python-e-django
https://tutorial.djangogirls.org/pt/
http://www.django-rest-framework.org/
http://imasters.com.br/desenvolvimento/definicao-restricoes-e-beneficios-modelo-de-arquitetura-rest/?trace=1519021197&source=single
http://www.rodrigocalado.com.br/o-que-e-rest-um-resumo-do-assunto-caracteristicas-conceitos-vantagens-e-desvantagens-prefiro-dizer-que-e-uma-rapida-introducao-ao-assunto/
https://www.infoq.com/br/articles/rest-soap-when-to-use-each
https://msdn.microsoft.com/pt-br/library/ms233836(v=vs.90).aspx
http://www.devmedia.com.br/web-services-rest-versus-soap/32451

https://www.infoq.com/br/articles/rest-soap-when-to-use-each

http://web.unipar.br/~seinpar/2014/artigos/pos/Cleber_de_F_Ferreira_Roberto_Dias_Mota%20(1).pdf

https://www.slideshare.net/nnja/djangocon-2014-django-rest-framework-so-easy-you-can-learn-it-in-25-minutes

https://www.slideshare.net/MarcelChastain/rest-easy-with-djangorestframework