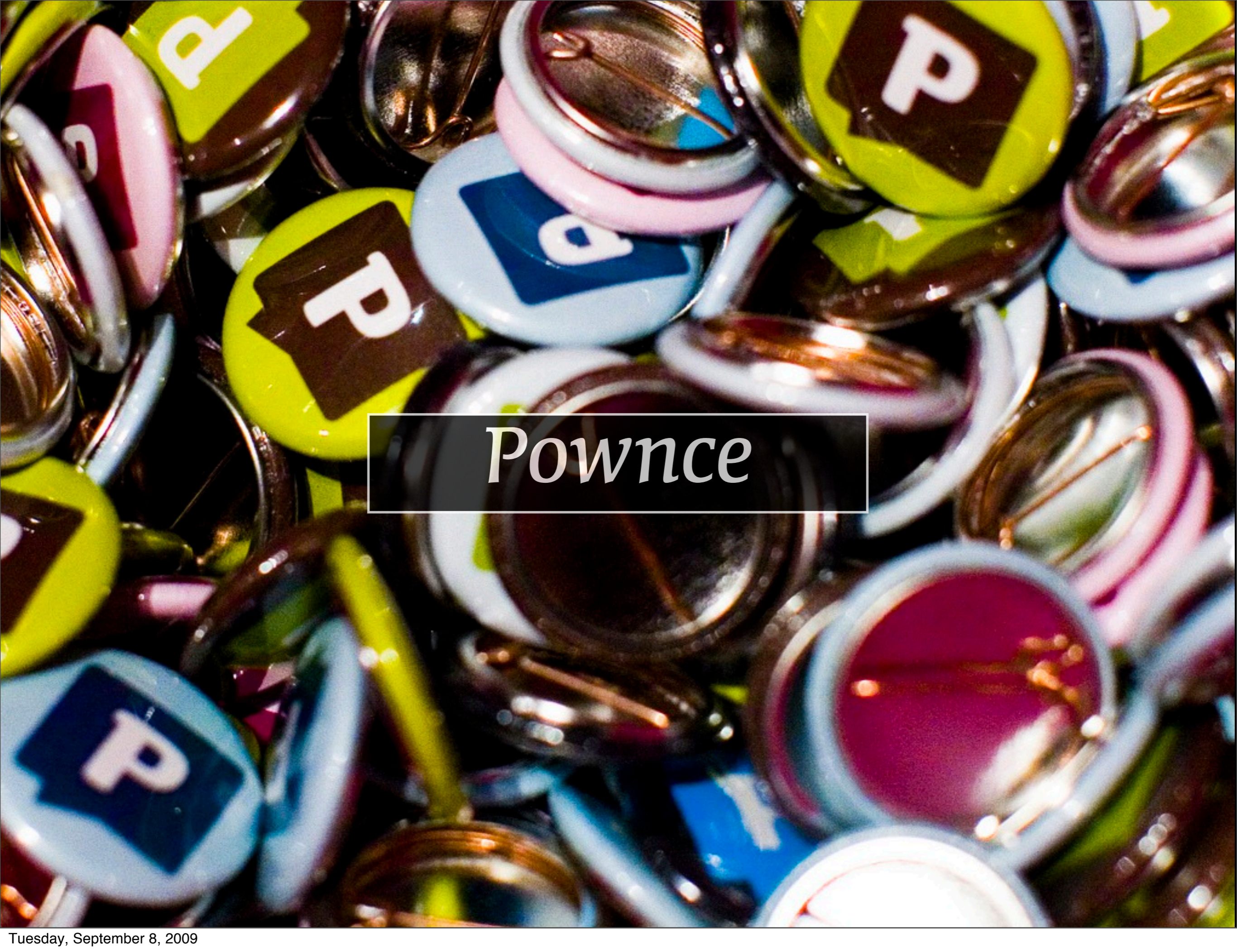


RESTful Ponies

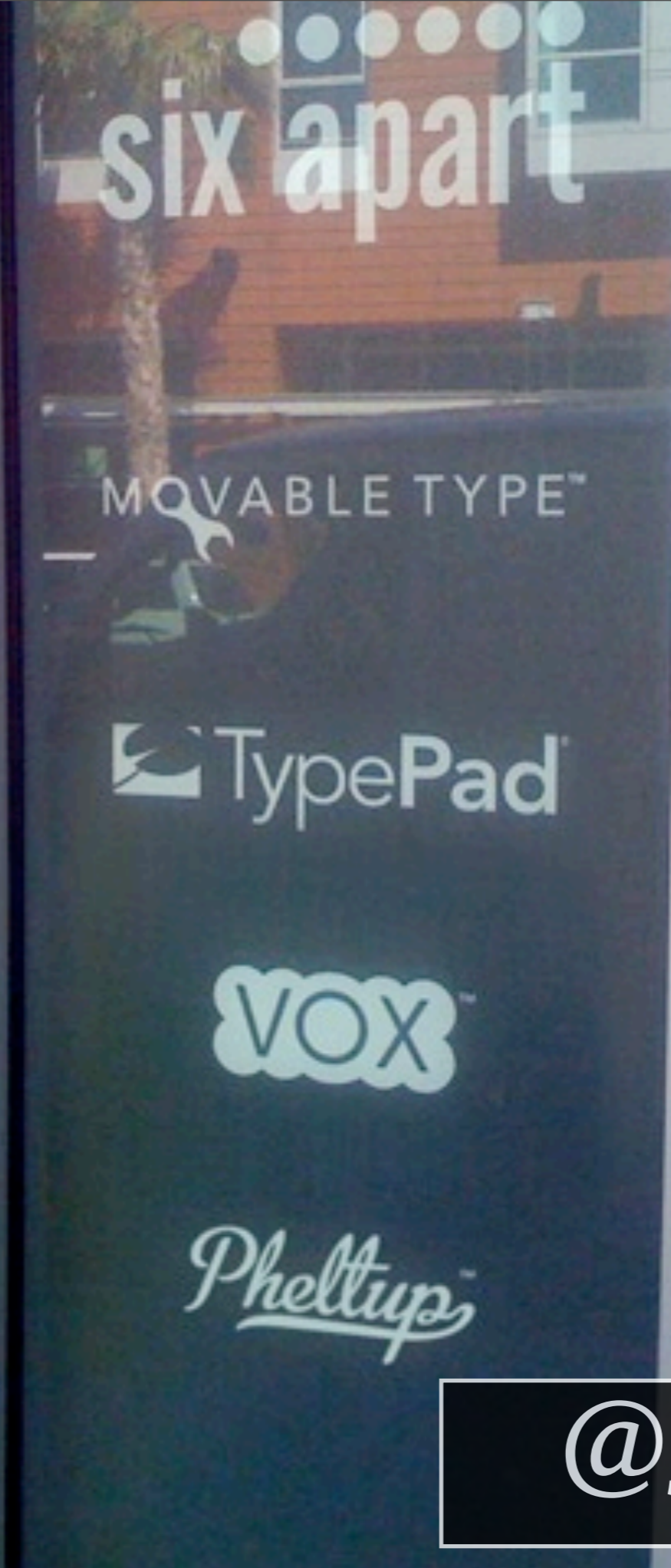
featuring Mike Malone



@mjmalone



Pownce



@sixapart

The Plan

1. **REST:** *what is REST?*
2. **RESTful Django:** *building RESTful services in Django*
3. **REST clients:** *talking to RESTful services*

REST

“REST is largely defined by a small number of semi-formal specifications [...] there are a number of contradictions, a number of ambiguities, and perhaps at least some assertions and principles which are not sufficiently supported.”

- Jeff Bone

REST

REST

- Dude named Roy T. Fielding, late 1999
 - Finishing up work on RFC 2616 - HTTP/1.1
 - Submitted his doctoral dissertation, entitled *Architectural Styles and the Design of Network-based Software Architectures*
 - Chapter 5: *Representational State Transfer (REST)*

Fielding Dissertation

“This dissertation defines a framework for understanding software architecture via architectural styles..”

Fielding Dissertation

*“An **architectural style** is a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style.”*

Architectural Style

An **architectural style** classifies an architecture.

A building has a specific architecture all its own, but it also may exhibit a particular *architectural style* like *Bauhaus, Baroque, or Functionalist*.

Architectural Style

The architecture of the *World Wide Web*, based on **HTTP**, conforms to the **REST** architectural style.

Architectural Style

Architectural Styles

are kinda like

Programming Paradigms

Architectural Style

Representational State Transfer

is kinda like

Object-oriented Programming

Architectural Style

- Object-oriented Programming
 - *A language* can be “object-oriented,” but you can still use it to write non-OO code
 - *An application* can be “object-oriented” even if the language it’s written in isn’t

Architectural Style

- The *Web* is RESTful, but you can still write *web services* that aren't
- Similarly, you could implement a RESTful architecture on top of an architecture that isn't RESTful
 - In fact, that's exactly what the Web does - TCP/IP isn't RESTful

RESTocity

- **Uniform Interface**

- Identification of resources
- Manipulation of resources through representations
- Self-descriptive messages
- Hypermedia as the engine of application state

- **Statelessness**

Wha-huh?

Let's go over some terminology...

Resource

A **Resource** is something that can be named



<http://flic.kr/p/6ky6FR>

Sign up

Please fill in all of these fields. Thanks!

Choose a unique username

Check

Password

First name

Last name

Country



Postal code or ZIP (Required for US, UK, and Canada)

Birthday

Month Day Year

Don't display my age

Gender

Select One

By signing up for Pownce you are agreeing to the terms of service.

Okay done

<http://flic.kr/p/27x8Xt>

Source Browser

Commits

Wiki

Network (1)

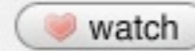
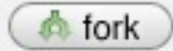
Watchers (2)

master

all branches

all tags

signup / free



😊 this repo is viewable by everyone

Clone URL: [git://github.com/signup/free.git](https://github.com/signup/free.git)

THE TURKS DID IT (take 3)



elliottcable (author)
21 minutes ago

commit 880fd34d55e993e5923cde787e69388c1abb5557
tree 4fe7ab6c5306286dc6f2f012a52a604b38004679
parent 19cae62b6c7e060b17f22e339deedcfd6c762612

free /

name	age	message	history
README.mkdn	20 minutes ago	THE TURKS DID IT (take 3) [elliottcable]	

README.mkdn

```
THE TURKS DID IT!
=====

No, really. They did. I lie not!

Anyway, this is bugged. You can probably reach the ACTUAL signup page here:

<http://github.com/signup/<the\_size\_you\_want/tree>>

(Don't ask *me* why, I have somewhere between 0 and 0 idea why...)
```

<http://flic.kr/p/4FTKJ4>





<http://flic.kr/p/2i3f1T>



<http://flic.kr/p/5VfEP>



@alex_gaynor

Resource

- A *resource* is just a *thing that can be named*
 - Information: *image, article, user profile*
 - Abstract: *current time, current weather*
- If a particular *resource* is interesting in your application, you should give that resource a *resource identifier*
 - For example, a *Universal Resource Identifier* or *URI*

Representation

- But you can't send the *resource* itself - you have to send a *representation*
 - A *representation* is a sequence of bytes, plus some *metadata*, that can be transferred between components
 - A single *resource* may have any number of *representations*

Hypermedia

A medium of information that contains hyperlinks

Application State

- State that affects how a request is processed
 - The query parameter in the URL for Google search results is an example of application state
- Not the same as *resource state* which is the current state of resources on the server, and is the same for all clients

RESTful Django

RESTful Web Services

- More than just applying the REST constraints:
 - RESTful *web services* should use the *uniform interface* implemented by the World Wide Web - HTTP
 - HTTP 1.1 is defined in **RFC 2616**
 - Defines all of the HTTP methods, status codes, and headers
 - Read it! It's really not that bad

Uniform Interface

Identification of Resources

Give resources URIs

urls.py

```
from django.conf.urls.defaults import *
from simplerest.api.views import breeds_resource,
breed_resource

urlpatterns = patterns('',
    (r'breeds/$', breeds_resource),
    (r'breeds/(?P<id>\d+)/$', breed_resource),
)
```

Uniform Interface

Manipulation of Resources through Representations

Resources are transmitted between components using
representations

Simple REST View

```
from ponies.models import Breed
from django.http import HttpResponse
from django.shortcuts import get_object_or_404
from django.utils import simplejson as json

def breed_resource(request, id):
    breed = get_object_or_404(Breed, id=id)
    d = {'name': breed.name,
        'min_height': breed.min_height,
        'max_height': breed.max_height,
        'origin': breed.origin}
    j = json.dumps(d)
    return HttpResponse(j,
                        content_type='application/json')
```

Simple REST View

```
$ curl -i 127.0.0.1:8000/api/breed/1/  
HTTP/1.0 200 OK  
Date: Fri, 04 Sep 2009 22:31:23 GMT  
Server: WSGIServer/0.1 Python/2.6.1  
Content-Type: application/json
```

```
{"min_height": 50, "origin": "CA", "name": "Canadian  
Rustic Pony", "max_height": 54}
```

Uniform Interface

Self-descriptive Messages

Requests and responses are stateless, use standard methods, status codes, and media-types, and explicitly indicate cacheability

Standard Methods

- There are four methods used essentially everywhere
 - **GET**: Transfer representation to the client
 - **POST**: Create a new resource in a collection
 - **PUT**: Create or updates specified resource
 - **DELETE**: Remove specified resource

Standard Methods

- Other methods
 - **HEAD**: Give me the message metadata (no content)
 - **OPTIONS**: What methods does this resource support
- There's a proposal to add a **PATCH** method that would work like **PUT** but you'd send a diff instead of the whole representation

0 noes...

```
$ curl -v -X DELETE 127.0.0.1:8000/api/breed/1/
> DELETE /api/breed/1/ HTTP/1.1
> User-Agent: curl/7.19.4 (i386-apple-darwin9.6.0)
> Host: 127.0.0.1:8000
> Accept: */*
>
< HTTP/1.0 200 OK
< Date: Fri, 04 Sep 2009 22:35:10 GMT
< Server: WSGIServer/0.1 Python/2.6.1
< Content-Type: application/json
<
{"min_height": 50, "origin": "CA", "name": "Canadian
Rustic Pony", "max_height": 54}
```

Less Simple REST View

```
from django.http import HttpResponseRedirect

def get_breed(request, id):
    if request.method == 'GET':
        # Return the Breed
    elif request.method == 'PUT':
        # Deserialize message and update the Breed
    elif request.method == 'DELETE':
        # Delete the Breed
    else:
        methods = ('GET', 'PUT', 'DELETE')
        response = HttpResponseRedirect(methods)
    return response
```

Denied.

```
$ curl -v -X POST 127.0.0.1:8000/api/breed/1/  
> POST /api/breed/1/ HTTP/1.1  
> User-Agent: curl/7.19.4 (i386-apple-darwin9.6.0)  
> Host: 127.0.0.1:8000  
> Accept: */*  
>  
< HTTP/1.0 405 METHOD NOT ALLOWED  
< Date: Fri, 04 Sep 2009 22:57:46 GMT  
< Server: WSGIServer/0.1 Python/2.6.1  
< Content-Type: text/html; charset=utf-8  
< Allow: GET, PUT, DELETE  
<
```

Status Codes

- Status codes are part of HTTP's uniform interface
 - Request methods identify RESTful requests
 - Status codes identify RESTful responses
- They're all defined in **RFC 2616**

Media Types

- The media type of a request or response is specified in the **Content-Type** HTTP header
- One of the most contentious topics in REST community
 - REST says nothing about the structure of representations
 - There must be a common understanding of the representation format between the client and server

Media Types

- Several popular formats are available
 - XML: Atom & extensions, application specific
 - RDF: FOAF
 - XHTML + microformats
 - JSON
- Try to find a standard format before inventing your own

Cacheability

- You can specify cacheability using the **Cache-Control** HTTP header
- Cache validators are used to check whether a cached item is still good
 - **Last-Modified** header indicates when the request was modified
 - **ETag** (entity tag) header is an opaque cache validator

Conditional GET

- Client specifies a cache validation header
 - **If-None-Match** for ETags
 - **If-Modified-Since** for Last-Modified
- If the condition is not met (e.g., the resource hasn't been updated and the cached representation is still valid) the server should return **304 NOT MODIFIED** with no content

Conditional GET

- Django has Conditional GET middleware

```
MIDDLEWARE_CLASSES += (  
    'django.middleware.http.ConditionalGetMiddleware',  
)
```

- But you still have to calculate ETag / Last-Modified header yourself

ETag

```
import hashlib
```

```
def theview(request):
```

```
    ...
```

```
    etag = hashlib.md5(response.content).hexdigest()
```

```
    response['ETag'] = "%s" % etag
```

```
    return response
```

Preconditions

- HTTP uses optimistic locking to prevent PUTs and DELETES from clobbering one another
- Relies on ETags and Last-Modified like Conditional GET
 - Client sends **If-Match** or **If-Unmodified-Since** header
 - If the ETags don't match, or the modification time is later than the client specified, a change has happened since the client fetched the resource and the server should return **412 PRECONDITION FAILED**

Preconditions

```
def breed_resource(request, id):
    if request.method == 'GET':
        ...
    elif request.method == 'PUT':
        breed = get_object_or_404(Breed, id=id)
        if not breed.check_precondition(request):
            response = HttpResponse(status=412)
        else:
            d = json.loads(request.raw_post_data)
            response = HttpResponse(json.dumps(d),
                                   content_type='application/json')
            etag = '"%s"' % md5.new(j).hexdigest()
            response['ETag'] = etag
    elif request.method == 'DELETE':
        breed = get_object_or_404(Breed, id=id)
        if not breed.check_precondition(request):
            response = HttpResponse(status=412)
        else:
            breed.delete()
            response = HttpResponse(status=204)
    ...
    return response
```

Uniform Interface

Hypermedia as the Engine of Application State (HATEOAS)

A representation should connect to related resources (i.e., related states) using hypermedia - links and forms - just like resources are connected on the human Web

HATEOAS

If a client follows a link in a **representation**, the client is **transferred** to a new **state**.

Hence the term **Representational State Transfer**.

Hypermedia

- Er, but JSON is a data serialization language, and doesn't define any mechanisms for hyperlinks or forms
- So we need to design a way to **link** to related resources in a JSON document

Designing Links

- More than just a URI, links typically include
 - The **URI** of the related (linked-to) resource
 - The **relationship** to the current resource
 - The **content-type** of the related resource
- And may also include
 - The **methods** supported by the related resource

JSON Link

```
{  
  "object-type": "tag:example.com,2009:Link",  
  "rel": "self",  
  "href": "http://example.com/ponies/42",  
  "type": "application/json",  
  "allowed-methods": ["GET", "PUT", "DELETE"]  
}
```

HATEOAS Pony

```
{
  "object-type": "tag:example.com,2009:Pony",
  "min_height": 48,
  "origin": "CA",
  "links": [
    {
      "object-type": "tag:example.com,2009:Link",
      "allowed-methods": ["GET", "PUT", "DELETE"],
      "href": "http://example.com/api/breeds/42/",
      "type": "application/json",
      "rel": "self"
    },
    {
      "object-type": "tag:example.com,2009:Link",
      "allowed-methods": ["GET", "POST"],
      "href": "http://example.com/api/breeds/",
      "type": "application/json",
      "rel": "collection"
    }
  ],
  "name": "Canadian Rustic Pony",
  "max_height": 54
}
```

HATEOAS Ponies

```
{
  "object-type": "tag:example.com,2009:Collection"
  "start-index": 0,
  "max-results": 25,
  "total-results": 102,
  "links": [
    {
      "object-type": "tag:example.com,2009:Link",
      "allowed-methods": ["GET", "POST"],
      "href": "http://example.com/api/ponies/",
      "type": "application/json",
      "rel": "self"
    }
  ],
  "entries": [
    {
      "object-type": "tag:example.com,2009:Link",
      "allowed-methods": ["GET", "PUT", "DELETE"],
      "href": "http://example.com/api/ponies/1/",
      "type": "application/json",
      "rel": "self"
    },
    ...
  ]
}
```

URIs are IDs

- Don't return database IDs or some other unique identifier instead of URIs
- Common complaint:
 - *“But if I use URIs as IDs I end up duplicating my base URI over and over again and generate big bloated responses”*

GZip is your friend

- If you're compressing your response documents, repeating strings aren't that big a deal

```
>>> url = 'http://example.com/api/breeds/'
>>> len(json.dumps([url] + [i for i in xrange(100)]))
422
>>> len(json.dumps([url + str(i) for i in xrange(100)]))
3490
>>> len(json.dumps([url] + [i for i in
                    xrange(100)]).encode('zlib'))
219
>>> len(json.dumps([url + str(i) for i in
                    xrange(100)]).encode('zlib'))
279
```

Sub-resources

- REST encourages linking to resources rather than including sub-resources
 - If you do include sub-resources, make sure clients can determine the resource type, and include a link rel=self
 - Problem: doing a lot of requests takes a while!
 - HTTP Pipelining helps with this, but it's not supported by many Web components
 - Keep this in mind, I'll get back to it a bit later

Statelessness

- Every HTTP request contains all of the information necessary for the server to fulfill the request
- All possible states of the application are also resources, and should be given their own URIs
- Application state should stay on the client, and should be transmitted to the server with each request (e.g., as headers or URI parameters)

Statelessness

Don't use sessions or cookies.

REST View

```
from django.http import HttpResponseRedirectNotAllowed

def get_breed(request, id):
    if request.method == 'GET':
        # Return the Breed
    elif request.method == 'PUT':
        # Deserialize message and update the Breed
    elif request.method == 'DELETE':
        # Delete the Breed
    else:
        methods = ('GET', 'PUT', 'DELETE')
        response = HttpResponseRedirectNotAllowed(methods)
    return response
```

PUT Method

```
if not authenticated(request):
    response = HttpResponse(status=401) # Unauthorized
    response['WWW-Authenticate'] = authenticate_headers(request)
    return response
if not authorized(request):
    return HttpResponse(status=403) # Forbidden
object = get_object_or_404(Object, id=id)
if not can_understand(request.META['CONTENT_TYPE']):
    return HttpResponse(status=415) # unsupported media type
try:
    decoded = json.loads(request.raw_post_data)
except ValueError, e:
    return HttpResponse(content=str(e), status=400) # bad request
if not validate(decoded):
    return HttpResponse(content='invalid attr value' status=400)
object.update_from_dict(decoded)
return HttpResponse(json.dumps(object.to_dict()),
                    content_type='application/json')
```

That's a lot of work...

- The whole view-as-a-function thing is a bit limiting when you're trying to expose Resources through Django
- Let's what are the real requirements for a Django view?
 - Must be **callable**
 - Must accept an **HttpRequest** as an argument
 - Must return an **HttpResponse**

RestView

```
from django.http import HttpResponseRedirect

class Resource(object):
    methods = ('get',)

    @classmethod
    def dispatch(cls, request, *args, **kwargs):
        handler = cls()
        method = request.method.lower()
        if method not in handler.methods:
            return HttpResponseRedirect(handler.methods)
        try:
            view = getattr(handler, method)
        except AttributeError:
            raise Exception("Missing method `%s`" % method)
        if not callable(view):
            raise Exception("Method `%s` uncallable" % method)
        return view(request, *args, **kwargs)
```

BreedResource

```
class BreedResource(RestView):
    content_type = 'application/json'

    def get_object(self, id):
        return get_object_or_404(Breed, id=id)

    def to_dict(self, breed):
        return {
            'name': breed.name,
            'min_height': breed.min_height,
            'max_height': breed.max_height,
            'origin': breed.origin
        }

    def etag(self, content):
        return '"%s"' % md5.new(content).hexdigest()

    def get(self, request, id):
        breed = self.get_object(id)
        j = json.dumps(self.to_dict(breed))
        response = HttpResponse(j, content_type=self.content_type)
        response['ETag'] = self.etag(j)
        return response
```

urls.py

```
from django.conf.urls.defaults import *
from api.resources import BreedResource

urlpatterns = patterns('',
    url(r'^breed/(?P<id>\d+)/$', BreedResource.dispatch),
)
```

django-piston

- Written by Jesper Noehr (@jespern) - creator of bitbucket
- Reusable components that work with Django to make creating REST web services easier
- Provides a mechanism for defining *resources* (separately from Django models) and creating views that handle all of the HTTP verbs
- Supports several serialization formats out of the box

django-piston

- Pluggable authentication & authorization system
 - Support OAuth, HTTP Digest & Basic out of the box
- Pluggable serialization architecture, allows you to send representations in a variety of content types
- Deserializes PUT and POSTed representations into request.data dictionary

piston Handler

```
from piston.handler import BaseHandler
from piston.utils import rc
from ponies.models import Breed

class BreedHandler(BaseHandler):
    allowed_methods = ('GET', 'PUT', 'DELETE')
    model = Breed

    def read(self, request, id):
        return Breed.objects.get(pk=id)

    def update(self, request, id):
        breed = Breed.objects.get(pk=id)
        breed.name = request.data.get('name')
        breed.max_height = request.data.get('max_height')
        breed.min_height = request.data.get('min_height')
        breed.origin = request.data.get('origin')
        breed.save()
        return breed

    def delete(self, request, id):
        post = Breed.objects.get(pk=id)
        post.delete()
        return rc.DELETED
```

piston urls.py

```
from django.conf.urls.defaults import *
from piston.resource import Resource
from piston.authentication import
HttpBasicAuthentication

from api.handlers import BreedHandler

breed_resource = Resource(BreedHandler)

urlpatterns = patterns('',
    url(r'^breed/(?P<id>\d+)/$', breed_resource),
)
```

PUT Request

```
$ curl -i -X PUT -H "Content-Type: application/json" -d  
"{\"min_height\": 48, \"origin\": \"CA\", \"name\":  
\"Canadian Rustic Pony\", \"max_height\": 60}" http://  
127.0.0.1:8000/api/breed/1/
```

```
HTTP/1.0 200 OK
```

```
Date: Sun, 06 Sep 2009 00:57:16 GMT
```

```
Server: WSGIServer/0.1 Python/2.6.1
```

```
Vary: Authorization
```

```
Content-Type: application/json; charset=utf-8
```

```
{  
  "min_height": 48,  
  "origin": "CA",  
  "name": "Canadian Rustic Pony",  
  "max_height": 60  
}
```

django-piston

<http://bitbucket.org/jespern/django-piston/>

django-roa

- Built on top of django-piston, automatically creates a RESTful API exposing your Django models as Resources
 - Similar to ActiveRecord in Rails
- Useful for simple APIs, and for building internal services
- You can even use it to make objects exposed through a remote service work with the Django admin

django-roa

<http://code.welldev.org/django-roa/>

batchhttp

- We've been playing around with using MIME to multiplex multiple HTTP requests within a single request
- The RESTfulness of this technique is questionable... but sometimes you just gotta be pragmatic
- We've even got a specification for it, written by Martin Atkins (@apparentlymart)

batchhttp

```
GET /api/breeds/1/ HTTP/1.1
```

```
User-Agent: curl/7.19.4 (i386-apple-darwin9.6.0) libcurl/  
7.19.4 zlib/1.2.3
```

```
Host: 127.0.0.1:8000
```

```
Accept: */*
```

```
GET /api/breeds/2/ HTTP/1.1
```

```
User-Agent: curl/7.19.4 (i386-apple-darwin9.6.0) libcurl/  
7.19.4 zlib/1.2.3
```

```
Host: 127.0.0.1:8000
```

```
Accept: */*
```

```
GET /api/breeds/3/ HTTP/1.1
```

```
User-Agent: curl/7.19.4 (i386-apple-darwin9.6.0) libcurl/  
7.19.4 zlib/1.2.3
```

```
Host: 127.0.0.1:8000
```

```
Accept: */*
```

batchhttp

```
POST /batch-processor
User-Agent: curl/7.19.4 (i386-apple-darwin9.6.0) libcurl/7.19.4 zlib/1.2.3
Host: 127.0.0.1:8000
Accept: */*
MIME-Version: 1.0
Content-Type: multipart/parallel;
            boundary="====5007185810729514936=="
```

HTTP MIME Message

```
--====5007185810729514936==
MIME-Version: 1.0
Content-Type: application/http-request
Multipart-Request-ID: 0
Content-transfer-encoding: quoted-printable
```

```
GET /api/breeds/1/ HTTP/1.1
User-Agent: curl/7.19.4 (i386-apple-darwin9.6.0) libcurl/7.19.4 zlib/1.2.3
Host: 127.0.0.1:8000
Accept: */*
```

```
--====5007185810729514936==
```

...

batchhttp library

- Low-level library for creating and parsing batch requests
- Batch request client built on the low-level library and *httplib2*
- Reverse proxy server built on *twisted* that parses batch requests and forwards them to a backend server

batchhttp

Specification: <http://bit.ly/batchhttp>

Python Library: <http://github.com/sixapart/batchhttp/>

REST Clients

HTTP Libraries

- The Python standard library comes with two HTTP libraries
 - urllib2
 - httplib
- Both are missing important features that we'll need to build a robust client

httplib2

- Developed by Joe Gregorio
- Sports a number of features that are useful for talking to web services
 - Caching
 - Compression
 - Extensible authentication mechanism

httplib2 GET

```
>>> import httplib2
>>> import json
>>> h = httplib2.Http()
>>> url = 'http://twitter.com/statuses/public_timeline.json'
>>> resp, content = h.request(url)
>>> l = json.loads(content)
>>> [i['user']['screen_name'] for i in l]
[u'zapperdink', u'therealcherilyn', u'hubat1',
u'EilishhaaG', u'CanelaEscucha', u'PrincesssTete',
u'kaitlynmarisa', u'takaakis62', u'Nelson_Tiago',
u'kierondonoghue', u'marcelo627', u'Sexy_Bekah',
u'binary42', u'chaceinfinite', u'AndrewTheLimey', u'satoru',
u'iMdaQuEeN87', u'LeeSOttawa', u'_vinnie', u'Bearly_Civil']
```

httplib2 POST

```
>>> h.add_credentials('101010', 'stillnotmypassword')
>>> url = 'http://twitter.com/statuses/update.json'
>>> headers = {'Content-Type': 'application/x-www-form-urlencoded'}
>>> body = urllib.urlencode({'status': "I like scotch."})
>>> resp, content = h.request(url, 'POST', headers=headers,
                             body=body)

>>> resp['status']
'200'
>>> d = json.loads(content)
>>> d['text']
u'I like scotch.'
```

httpplib2

<http://code.google.com/p/httpplib2/>

OAuth with httpplib2

<http://bit.ly/httpplib2-oauth>

thanks @markpasc

Generic REST Client

- You should be able to talk to a truly RESTful web service using a generic client (you know, like a web browser)
 - Unfortunately, in the real world there are very few truly RESTful web services
 - And we can't agree on stuff like content types, so we're not quite there yet from a standardization perspective
- We're left with client libraries targeting individual services

remoteobjects

- We work a lot with RESTish web services at Six Apart
- Mark Paschal (@markpasc) created an abstraction layer that allows us to create robust client libraries very easily
 - The remoteobjects library allows you to describe an API using a declarative syntax, like Django models
 - Works very well with RESTful APIs, but it's generic enough to work with RPC APIs too

remoteobjects

```
class User(RemoteObject):
    id = fields.Field()
    name = fields.Field()
    screen_name = fields.Field()
    [ ... ]
    followers_count = fields.Field()
    status = fields.Object('status')

    @classmethod
    def get_user(cls, http=None, **kwargs):
        url = '/usrs/show'
        if 'id' in kwargs:
            url += '/%s.json' % quote_plus(kwargs.pop('id'))
        else:
            url += '.json'
        query = urlencode(kwargs)
        url = urlunsplit((None, None, url, query, None))
        return cls.get(urljoin(Twitter.endpoint, url),
                        http=http)
```

remoteobjects

```
>>> t = twitter.Twitter()
>>> t.add_credentials('mjmalone', 'thisisnotmypassword')
>>> friends_timeline = t.friends_timeline()
>>> [e.user.screen_name for e in friends.entries]
['ronaldheft', 'joestump', 'donttrythis',
'laughingsquid', 'FrankGruber', 'courtstarr', 'mbaratz',
'djchall', 'dozba', 'chrismessina', 'pop17', 'ijustine',
'calden', 'bryanveloso', 'jessenoller', 'pierre',
'optimarcusprime', 'snackfight', 'shiralazar']
```

remoteobjects

<http://github.com/sixapart/remoteobjects>

```
<link rel='self' />
```

http://immike.net/files/restful_ponies.pdf

Resources

Fielding Dissertation: <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

RFC 2616: <http://www.w3.org/Protocols/rfc2616/rfc2616.html>

REST Wiki: <http://rest.blueoxen.net/>

REST Worst Practices: <http://jacobian.org/writing/rest-worst-practices/>

How I Explained REST to my Wife: <http://tomayko.com/writings/rest-to-my-wife>

Questions?

Contact me @mjmalone or mjmalone@gmail.com