Django-piston

creating restful/resource oriented
webservices using Django

Representational state transfer

Set of **design criteria** for building
distributed hypermedia systems inspired
by the **principles** that made the
World Wide Web succesful

Principles

- Client-server
- Stateless
- Cacheable
- Layered system
- Uniform interface
  - Identification of resources
  - Manipulation of resources through these representations
  - Self-descriptive messages
  - Hypermedia as the engine of application state

Resource Oriented Architecture

A **RESTful architecture** for designing **web services**

Architecture

- Tied to HTTP
- Uniform interface
- Addressability
  - clean, meaningful, well structured
- Safety
- Idempotence
- Connectedness
- Statelesness
  - No state means scalable and reliable

A resource

*"A **resource** is **anything** important enough to be **referenced** as a **thing in itself**"*

-- RESTful Web Services, O'Reilly

Has a representation

Data format documenting current/intended **state** of the **resource**

Used to transfer state

Client is able to change **server state** by sending a **representation** of the new state of a **resource**.

Django-piston

"A **mini-framework** for **Django** for creating **RESTful APIs**."

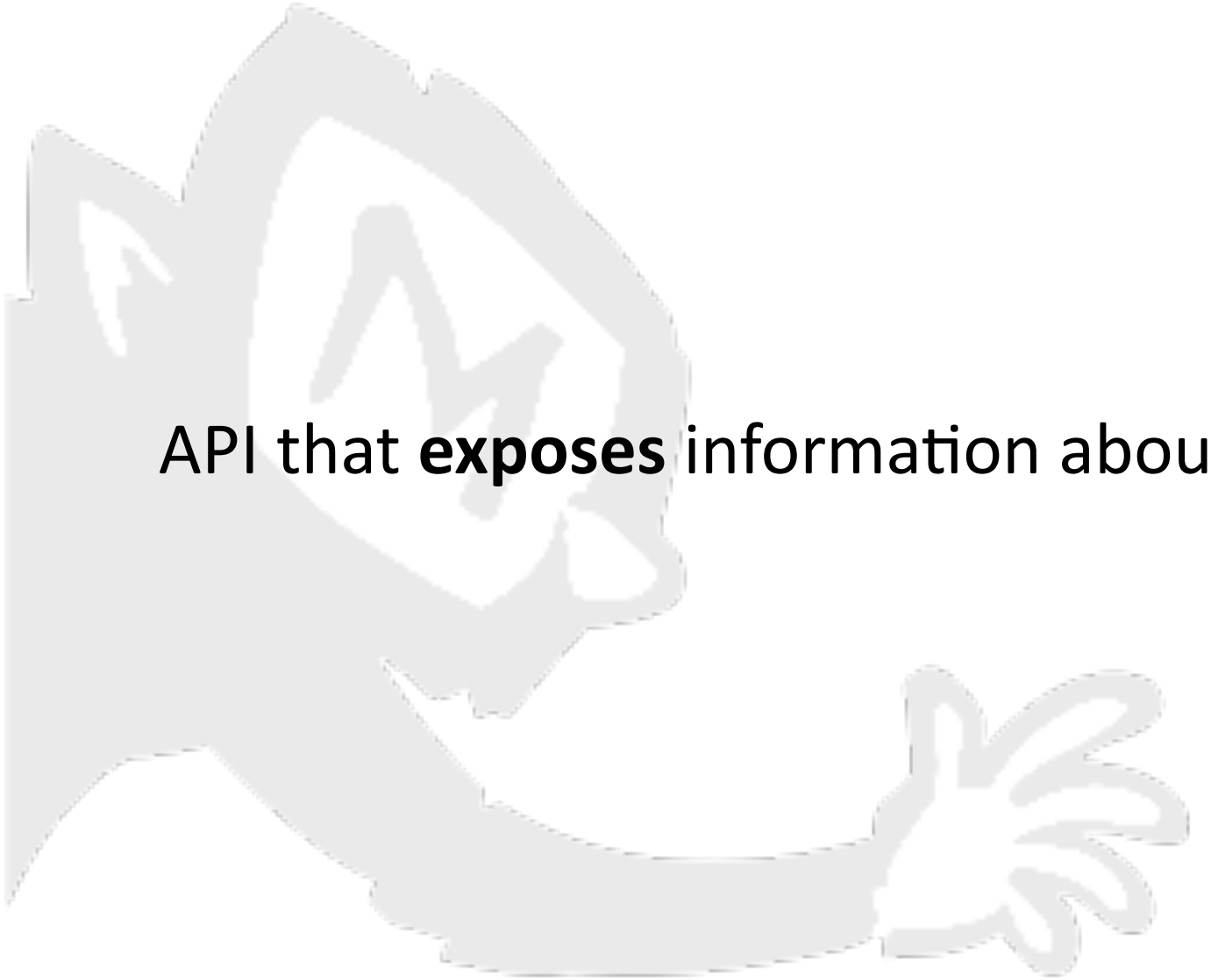
Features

- Ties into Django's internal mechanisms.
- Supports OAuth out of the box (as well as Basic/Digest or custom auth.)
- Doesn't require tying to models, allowing arbitrary resources.
- Speaks JSON, YAML, Python Pickle & XML

Features

- Ships with a convenient reusable library in Python
- Respects and encourages proper use of HTTP (status codes, ...)
- Has built in (optional) form validation (via Django), throttling, etc.
- Supports streaming, with a small memory footprint.
- Stays out of your way.

Build an API in 5 minutes! ;)



API that **exposes** information about **countries**

The model

```
# models.py
from django.db import models

class Country(models.Model):
    name = models.CharField(blank=False,
                            max_length=100, db_index=True)
    slug = AutoSlugField(populate_from='name')
```

Country overview

- Request to **/country/** will result in a JSON representation of all countries (the queryset)
- Request to **/country/1/** will result in returning only the country object with pk 1

Implementation

```
# handlers.py
from piston.handler import BaseHandler
from models import Country

class CountryHandler(BaseHandler):
  model = Country
  allowed_methods = ('GET',)

# urls.py
from django.conf.urls.defaults import patterns, url
from piston.resource import Resource
from handlers import CountryHandler

urlpatterns = patterns('',
  url(r'^country/(?P<pk>[^/]+)/$',
        Resource(CountryHandler)
  ),
)
```

Using slugs

- Request to **/country/** will result in a JSON representation of all countries (the queryset)
- Request to **/country/netherlands/** will result in returning only the country object with the slug field set to netherlands

Implementation

```
# handlers.py
class CountryHandler(BaseHandler):
    ...
    # Update pk to country_slug in urls.py
    def read(self, request, country_slug=None):
        if country_slug is not None:
            return get_object_or_404(Country,
                                     slug=country_slug)
        else:
            return Country.objects.all()
```

Updating countries

- PUT Request to **/country/1/** with all of the models variables will result in a change of the resource state (to the new desired state)

Implementation

```
# handlers.py
class CountryHandler(BaseHandler):
    model = Country
    allowed_methods = ('GET', 'PUT')

    ...
```

Validation of PUT

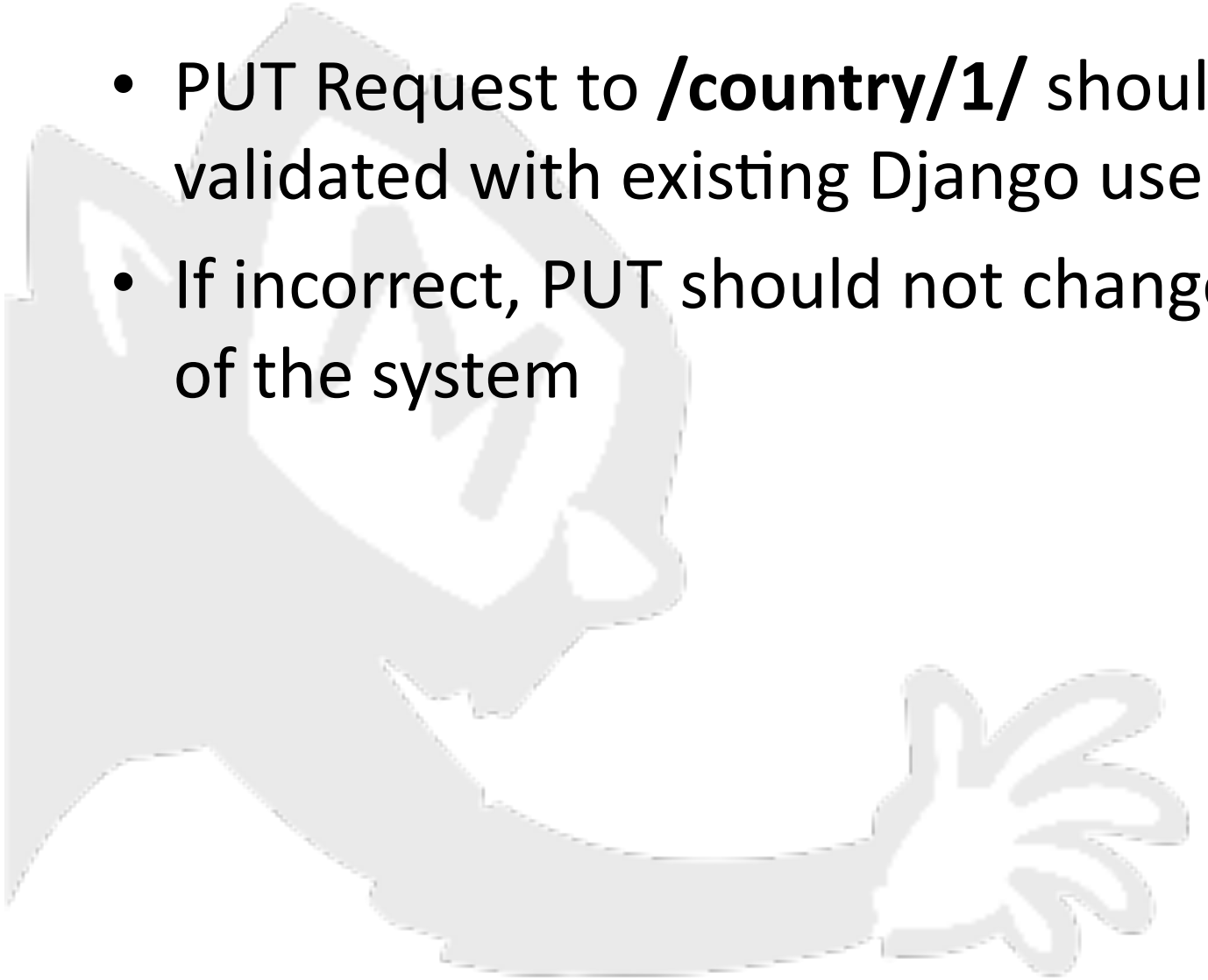
- PUT Request to **/country/1/** should be validated (and if incorrect, PUT should not change the state of the system)

Implementation

```
# handlers.py
from django import forms

class CountryForm(forms.ModelForm):
    class Meta:
        model = Country

class CountryHandler(BaseHandler):
  ..

  @validate(CountryForm, 'PUT')
  def update(self, request, pk):
    super(CountryHandler, self).update(request, pk)
```

Authentication

- PUT Request to **/country/1/** should be validated with existing Django users
- If incorrect, PUT should not change the state of the system

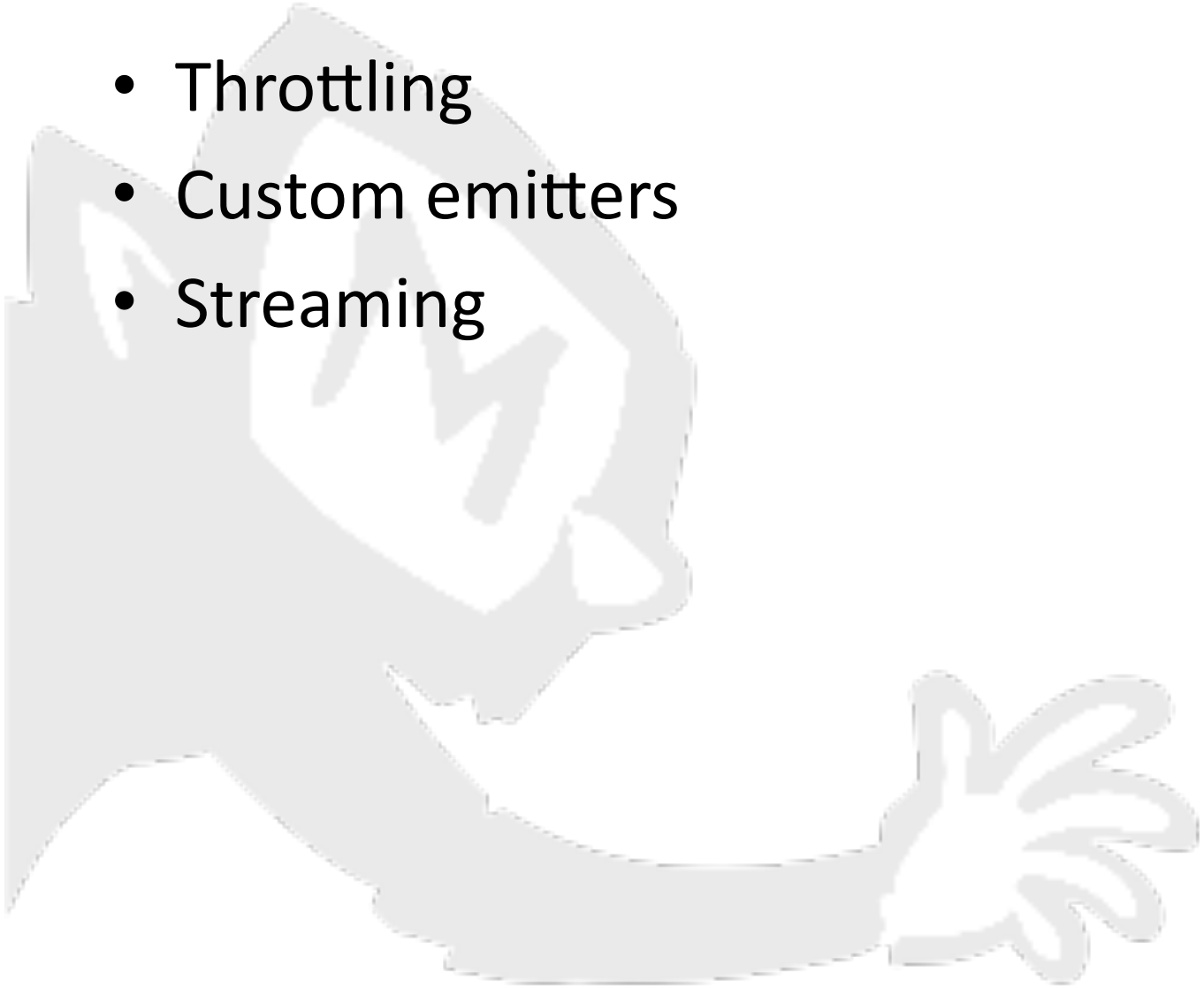


Implementation

```
# urls.py
from piston.authentication import HttpBasicAuthentication

basic_auth = \
      HttpBasicAuthentication(realm='CountryService')

urlpatterns = patterns('',
  url(r'^(?P<country>[^/]+)/$',
      Resource(CountryHandler,
                    authentication=basic_auth)
      ),
)
```

And more

- Throttling
- Custom emitters
- Streaming



Danki!

Questions?

Best practices

- versioning in your api
- using headers -> localization/content negotiation
- make a separate api namespace
- have each functional part of the api be it's own app (just like you would normally do with django)