django-wms Documentation

Release 0.1.2

Daniel Wiesmann

Apr 16, 2017

Contents

The Django WMS Framework is a toolkit that makes it easy to integrate a Web Map Service (WMS) or a x-y-z Tile Map Service into a Django project. Rendering of both vector and raster data formats are supported.

CHAPTER 1

Requirements

The processing of spatial data in django-wms relies on MapServer and its python bindings MapScript. Raster data integration depends on the django-raster package. The use of PostGIS as the database backend is required as well, for raster integration PostGIS >= 2.0 is required (see also django-raster package).

CHAPTER 2

Installation

1. Install package with `pip install django-wms`
2. Add "wms" to your INSTALLED_APPS setting like this

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

CHAPTER 3

Introduction

The structure of django-wms is closely tied to how MapServer works.

The spatial data rendering in django-wms relies on MapScript, the python-bindings for MapServer. The basic funcitonality of django-wms is to use mapscript to dynamically produce directives for MapServer. To render spatial data, MapServer is configured through MapFiles in which global WMS parameters, layers and cartograpy styles are defined.

The concept of MAP directives is translated into a `WmsMap` class. LAYER definitions and classes for symbology and cartography are represented in the `WmsLayer` class within django-wms. A set of SYMBOL directives for drawing point data are preconfigured as well.

MapScript has its own class definitions for those directives, django-wms simply makes them easy to use in a django project.

Web requests in django-wms are handled through a class based view module `WmsView`. The view can be hooked into a url and will take care of handling WMS and TMS requests automatically.

CHAPTER 4

Example

To create a mapping service, subclass the django-wms layer, map and view classes and connect them to an existing model in django that has a spatial field (such as Point, Polygon, MultiPolygon or Raster). An example `wms_config.py` module could be specified as follows

```
### wms_config.py

# Load django-wms classes
from wms import maps, layers, views

# Load model with spatial field (Point, Polygon, MultiPolygon)
from myapp.models import MySpatialModel

# Subclass the WmsVectorLayer class and point it to a spatial model.
# Use WmsRasterLayer for rasters
class MyWmsLayer(layers.WmsVectorLayer):
    model = MySpatialModel

# Subclass the WmsMap class and add the layer to it
class MyWmsMap(maps.WmsMap):
    layer_classes = [ MyWmsLayer ]

# Subclass the WmsView to create a view for the map
class MyWmsView(views.WmsView):
    map_class = MyWmsMap
```

With the WmsView subclass in place, the only thing left to do to create a functional map service is to hook the view into a url. An example url configuration `urls.py` could be

```
### urls.py

# Import the wms view
from myproject.wms_config import MyWmsView

# Add url patterns to setup map services from the view
urlpatterns = patterns('',
```

```
    # This creates a WMS endpoint
    url(r'^wms/$', MyWmsView.as_view(), name='wms'),

    # This creates a TMS endpoint
    url(r'^tile/(?P<layers>[^/]+)/(?P<z>[0-9]+)/(?P<x>[0-9]+)/(?P<y>[0-9]+)(?P<format>
↪\.jpg|\.png)$',
        MyWmsView.as_view(), name='tms'),
)
```

The django-wms package will automatically detect the first spatial field it can find in `MySpatialModel` and create a WMS endpoint from the class based view. If the three arguments `x`, `y` and `z` are found in the urlpattern, the view functions as TMS endpoint.