django-geojson Documentation

Release 2.10

Makina Corpus

Oct 17, 2022

Contents

Contents:

CHAPTER 1

Installation

```
pip install django-geojson
```

This package has **optional** extra dependencies.

If you need GeoJSON fields with map widgets :

```
pip install "django-geojson [field]"
```

1.1 Configuration

Add `djgeojson` to your applications :

```
# settings.py

INSTALLED_APPS += (
    'djgeojson',
)
```

(not required for views)

CHAPTER 2

Views

2.1 GeoJSON layer view

Very useful for web mapping :

```
# urls.py
from djgeojson.views import GeoJSONLayerView
...
url(r'^data.geojson$', GeoJSONLayerView.as_view(model=MushroomSpot), name='data'),
```

Consume the vector layer as usual, for example, with Leaflet loaded with jQuery Ajax:

```
# Leaflet JS
var layer = L.geoJson();
map.addLayer(layer);
$.getJSON("{% url 'data' %}", function (data) {
    layer.addData(data);
});
```

Inherit generic views **only** if you need a reusable set of options :

```
# views.py
from djgeojson.views import GeoJSONLayerView

class MapLayer(GeoJSONLayerView):
    # Options
    precision = 4   # float
    simplify = 0.5  # generalization


# urls.py
from .views import MapLayer, MeetingLayer
...
url(r'^mushrooms.geojson$', MapLayer.as_view(model=MushroomSpot, properties=('name',
↪)), name='mushrooms')
```

(continues on next page)

(continued from previous page)

Most common use-cases of reusable options are: low-fi precision, common list of fields between several views, etc.

Options are :

- **properties** : `list` of properties names, or `dict` for mapping field names and properties
- **simplify** : generalization of geometries (See `simplify()`)
- **precision** : number of digit after comma
- **geometry_field** : name of geometry field (*default*: `geom`)
- **srid** : projection (*default*: 4326, for WGS84)
- **bbox** : Allows you to set your own bounding box on feature collection level
- **bbox_auto** : True/False (default false). Will automatically generate a bounding box on a per feature level.
- **use_natural_keys** : serialize natural keys instead of primary keys (*default*: `False`)
- **with_modelname** : add the app and model name to the properties. (*default*: `True`)
- **crs_type** : add the type of crs generated, options: `name` and `link` (*default*: `name`)

2.2 Tiled GeoJSON layer view

Vectorial tiles can help display a great number of objects on the map, with reasonnable performance.

```
# urls.py
from djgeojson.views import TiledGeoJSONLayerView
...

url(r'^data/(?P<z>\d+)/(?P<x>\d+)/(?P<y>\d+).geojson$',
    TiledGeoJSONLayerView.as_view(model=MushroomSpot), name='data'),
```

Consume the vector tiles using Leaflet TileLayer GeoJSON, Polymaps, OpenLayers 3 or d3.js for example.

Options are :

- **trim_to_boundary** : if `True` geometries are trimmed to the tile boundary
- **simplifications** : a dict of simplification values by zoom level

2.3 GeoJSON template filter

Mainly useful to dump features in HTML output and bypass AJAX call :

```
// Leaflet JS
L.geoJson({{ object_list|geojsonfeature|safe}}).addTo(map);
```

Will work either for a model, a geometry field or a queryset.

```
{% load geojson_tags %}

var feature = {{ object|geojsonfeature|safe }};
```

(continues on next page)

(continued from previous page)

```
var geom = {{ object.geom|geojsonfeature|safe }};

var collection = {{ object_list|geojsonfeature|safe }};
```

Properties and custom geometry field name can be provided.

```
{{ object|geojsonfeature:"name,age" }}
{{ object|geojsonfeature:"name,age:the_geom" }}
{{ object|geojsonfeature:":geofield" }}
```

CHAPTER 3

Model and forms fields

GeoJSON fields are based on Django JSONField. See *Installation* to install extra dependencies.

They are useful to avoid usual GIS stacks (GEOS, GDAL, PostGIS...) for very simple use-cases (no spatial operation yet).

```
from djgeojson.fields import PointField

class Address(models.Model):
    geom = PointField()

address = Address()
address.geom = {'type': 'Point', 'coordinates': [0, 0]}
address.save()
```

Form widgets are rendered with Leaflet maps automatically if django-leaflet is available.

All geometry types are supported and respectively validated : *GeometryField*, *PointField*, *MultiPointField*, *LineStringField*, *MultiLineStringField*, *PolygonField*, *MultiPolygonField*, *GeometryCollectionField*.

CHAPTER 4

Advanced usage

4.1 Low-level serializer

```
from djgeojson.serializers import Serializer as GeoJSONSerializer

GeoJSONSerializer().serialize(Restaurants.objects.all(), use_natural_keys=True, with_
↪modelname=False)
```

4.2 Low-level deserializer

```
from djgeojson.serializers import Serializer as GeoJSONSerializer

GeoJSONSerializer().deserialize('geojson', my_geojson)
```

You can optionally specify the model name directly in the parameters:

```
GeoJSONSerializer().deserialize('geojson', my_geojson, model_name=my_model_name)
```

4.3 Dump GIS models, or fixtures

Register the serializer in your project :

```
# settings.py

SERIALIZATION_MODULES = {
    'geojson' : 'djgeojson.serializers'
}
```

Command-line `dumpdata` can export files, viewable in GIS software like QGis :

```
python manage.py dumpdata --format=geojson yourapp.Model > export.geojson
```

Works with `loaddata` as well, which can now import GeoJSON files.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search