django-geostore-routing

Release 0.9.7.dev0

Makina Corpus

Nov 06, 2020

CONTENTS

PGRouting plugin for django-geostore

CHAPTER

ONE

INSTALLATION

1.1 Requirements

Minimum configuration :

- Python 3.6+
- PostgreSQL 10+
- PostGIS 2.4+
- PgRouting 2.5+

Recommended configuration :

- Python 3.8
- PostgreSQL 12
- PostGIS 3
- PgRouting 3

Your final django project should use django.contrib.gis.backend.postgis as default DATABASE backend

USING docker image :

https://hub.docker.com/r/pgrouting/pgrouting

1.1.1 SYSTEM REQUIREMENTS

these are debian packages required

- libpq-dev (psycopg2)
- gettext (translations)
- binutils (django.contrib.gis)
- libproj-dev (django.contrib.gis)
- gdal-bin (django.contrib.gis)

recommended

- postgresql-client (if you want to use ./manage.py dbshell command)

1.2 With pip

From Pypi:

```
pip install django-geostore-routing
```

From Github:

```
pip install -e https://github.com/Terralego/django-geostore-routing.git@master
↪#egg=django-geostore-routing
```

1.3 With git

```
git clone https://github.com/Terralego/django-geostore-routing.git
cd django-geostore-routing
python setup.py install
```

1.4 In your project settings

```
INSTALLED_APPS = (
  ...
  "geostore",
  "geostore_routing",
  ...
)
```

1.5 Settings up

pgRouting needs to update a table that contains all linestring to create topological connections. You need to execute a command to create topology at first. Once, after every feature update topology will be automatically updated if you enable GEOSTORE_ROUTING_CELERY_ASYNC with a working celery worker.

CHAPTER

TWO

ROUTING API

django-geostore-routing integrate a way to use your LineString layer as a routing one.

2.1 Arguments

First attribute needed, and mandatory, is `geom`, it must contains a LineString from start to endpoint, passing through all the way points. Geostore will create a path passing on the intersection the closest of those point, in the order you provided it.

It can also be provided a `callbackid`, that is used to identify the request. It can be useful in async environment. The `callbackid` is provided «as is» in the response.

Query content can provided in a POST or a GET request.

An example of response:

```
{
    'request': {
        'callbackid': 'my_callback',
        'geom': {
            "type": 'LineString',
            "coordinates": [
            [
                10.8984375,
                52.1874047455997
            ],
            [
                1.58203125,
                46.042735653846506
            ]
            ]
        }
    },
    'geom': {
        'type': 'LineString',
        'coordinates': [
        [
            1.6259765625,
            45.767522962149876
        ],
        [
            5.2294921875,
            46.558860303117164
        ],
```

(continues on next page)

(continued from previous page)

```
        [
            10.986328125,
            52.10650519075632
        ]
        ]
    },
    'route': {
        "type": "FeatureCollection",
        "features": [
            {
                "type": "Feature",
                "geometry": {
                    'type': 'LineString',
                    'coordinates': [
                    [
                        1.6259765625,
                        45.767522962149876
                    ],
                    [
                        5.2294921875,
                        46.558860303117164
                    ]
                    ]
                },
                "properties": {
                   "id": 1
                 },
            },
            {
                "type": "Feature",
                "geometry": {
                    'type': 'LineString',
                    'coordinates': [
                    [
                        5.2294921875,
                        46.558860303117164
                    ],
                    [
                        10.986328125,
                        52.10650519075632
                    ]
                    ]
                },
                "properties": {
                   "id": 2
                 },
            }
        ]
    }
}
```

CHAPTER

THREE

COMMANDS

3.1 update_topology

```
./manage.py update_topology -pk <layer_pk> --tolerance <tolerance>
```

You must provide the pk of the layer you want to use. Tolerance for extremity snapping is 0.00001 by default (unity should match to your INTERNAL_GEOMETRY_SRID, by default for 4326 see https://www.usna.edu/Users/oceano/pguth/md_help/html/approx_equivalents.htm)

CHAPTER

FOUR

SETTINGS

4.1 GEOSTORE_ROUTING_CELERY_ASYNC

Default: False

Boolean that activate automatic topology update in celery worker. Use only if a celery working is activated and configured for your project. Until, use update_topology command

4.2 GEOSTORE_ROUTING_TOLERANCE

Default: 0.000001

Tolerance to snap geometries in topologies. This default value match with INTERNAL_GEOMETRY_SRID unit (default WGS84 4326, angles)