django-geoip Documentation

Release 0.2.3

coagulant

April 15, 2012

CONTENTS

App to figure out where your visitors are from by their IP address.

Note: Currentrly `django-geoip` supports only ipgeobase.ru backend. It provides accurate geolocation in Russia and Ukraine only. There are plans to add other backends in future releases.

CHAPTER

ONE

CONTENTS

1.1 Installation

This app works with python 2.6-2.7, Django 1.3 and higher.

Recommended way to install is pip:

```
pip install django-geoip
```

1.1.1 Basic

- Add `django_geoip` to `INSTALLED_APPS` in settings.py:

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

- Run `python manage.py syncdb` or `python manage.py migrate` (if you're using South)
- Run `python manage.py ipgeobase_update` to obtain latest IpGeoBase data.

1.1.2 Advanced

In order to make user's location detection automatic several other steps are required:

- Add `LocationMiddleware` to `MIDDLEWARE_CLASSES`:

```
MIDDLEWARE_CLASSES = (...
    'django_geoip.middleware.LocationMiddleware',
    ...
)
```

- Include app urls into your urlconf if you want to allow visitors to change their region:

```
urlpatterns += patterns('',
    ...
    (r'^geoip/', include('django_geoip.urls')),
    ...
)
```

- Provide a custom location model (inherited from django_geoip.models.GeoLocationFascade)

- Specify this model in settings:

```
GEOIP_LOCATION_MODEL = 'example.models.Location' #example
```

1.2 How it works

1.2.1 Data storage

All geoip data, including geograpy and geoip mapping is stored in the database.

Geography

Right now django-geoip supports only ipgeobase geography, which consist of following entities: Country, Region, City. Database maintains normalized relationships between all entities, i.e. Country has many Regions, Region has many Cities.

class `django_geoip.models.`**`Country`**(**args*, ***kwargs*)
One country per row, contains country code and country name (TBD, #2)

class `django_geoip.models.`**`Region`**(**args*, ***kwargs*)
Region is a some geographical entity that belongs to one Country, Cities belong to one specific Region. Identified by country and name.

class `django_geoip.models.`**`City`**(**args*, ***kwargs*)
Geopoint that belongs to the Region and Country. Identified by name and region. Contains additional latitude/longitude info.

IP ranges

class `django_geoip.models.`**`IpRange`**(**args*, ***kwargs*)
IP ranges are stored in separate table, one row for each ip range.

Each range might be associated with either country (for IP ranges outside of Russia and Ukraine) or country, region and city together.

Ip range borders are stored as long integers

1.2.2 Low-level API usage

Here is an example of how can you guess user's location:

```
from django_geoip.models import IpRange

ip = "212.49.98.48"

try:
    ipgeobases = IpRange.objects.by_ip(ip)
    print ipgeobase.city #    ()
    print ipgeobase.region #   ( )
    print ipgeobase.country #   ()
except IpRange.DoesNotExist:
    print u'Unknown location'
```

1.3 High-level API usage

The app provides a convenient way to detect user location automatically. If you've followed advanced installation instructions, you can access user's location in your `request` object:

```
def my_view(request):
    """ Passing location into template """
    ...
    context['location] = request.location
    ...
```

User location is an instance of a custom model that you're required to create on your own (details below).

To avoid unnecessary database hits user location id is stored in a cookie.

1.3.1 Location model

Location model suites the basic needs for sites with different content for users, depending on their location. Ipgeobase forces Country-Region-City geo-hierarchy, but it's usually too general and not sufficient. Site content might depend on city only, or vary on custom areas, combining various cities, that don't match actual geographic regions.

In order to abstract geography from business logic, django-geoip requires a model, specific to your own app.

Creating custom location model

Create a model, that inherits from `django_geoip.models.GeoLocationFascade`. It should implement following classmethods:

class `django_geoip.models.`**`GeoLocationFascade`**(**args*, ***kwargs*)

Interface for custom geographic models. Model represents a fascade pattern for concrete GeoIP models.

classmethod `get_available_locations`()

Return all locations available for users to select in frontend

Returns GeoLocationFascade

classmethod `get_by_ip_range`(*ip_range*)

Return single model instance for given IP range. If no location matches the range, raises DoesNotExist exception.

Parameters **ip_range** (*IpRange*) – User's IpRange to search for.

Returns GeoLocationFascade single object

classmethod `get_default_location`()

Return default location for cases where ip geolocation fails.

Returns GeoLocationFascade

1.3.2 Switching region

Works very much like The set_language redirect view. Make sure you've included `django_geoip.urls` in your urlpatterns. Note that `set_location` view accepts only POST requests.

1.4 Ipgeobase backend

ipgeobase.ru is a database of russian and ukranian IP networks mapped to geographical locations.

It's maintained by RuCenter and updated daily.

As of 9 April 2012 it contains info on 952 cities and 145736 Ip Ranges (some networks doesn't belong to CIS).

1.5 Updating GeoIP database

Note: Currentrly `django-geoip` supports only ipgeobase.ru backend.

To update your database with fresh entries (adds new geography and completely replaces all IpRegions with fresh ones):

```
python manage.py geoip_update
```

If you wish to clear all geodata prior the sync (deletes all Cities, Regions, Countries and IpRanges):

```
python manage.py geoip_update --clear
```

1.6 Settings

`django-geoip` has some public and internal configuration:

class `django_geoip.geoip_settings.`**`GeoIpConfig`**(***kwargs*)

GeoIP configuration

`COOKIE_DOMAIN` = ''

Fill in for custom case

`COOKIE_EXPIRES` = 31622400

Cookie lifetime in seconds (1 year by default)

`COOKIE_NAME` = 'geoip_location_id'

Cookie stores location model primary key

`LOCATION_MODEL` = 'django_geoip.models.GeoLocationFascade'

Provide a model that stores geography, specific to your application

`STORAGE_CLASS` = 'django_geoip.storage.LocationCookieStorage'

Persistant storage class for user location (cookies or dummy are available)

1.7 Reference

TBD

1.8 Changelog

1.8.1 0.2.3dev (2012-04-??)

- Added country names
- Management update command renamed from `ipgeobase_update` to `geoip_update`
- Management command verbose output with progressbar
- Dropped django 1.2 support
- Documentation improved

1.8.2 0.2.2 (2012-01-25)

- Fixed middleware behavior when preocess_request never ran (redirects)
- Improved location storage validation, fixed cookie domain detection
- Added `Locator.is_store_empty` function to reveal if geoip detection was made

1.8.3 0.2.1 (2012-01-25)

- Fixed middleware behavior when request.location is None
- Added `GEOIP_STORAGE_CLASS` setting to override default user location storage
- Introduced `LocationDummyStorage` class to avoid cookie storage

1.8.4 0.2 (2012-01-20)

- Major refactoring of the app, added more tests
- Fixed a typo in `get_availabe_locations`

1.8.5 0.1 (2012-01-18)

- Initial release

CHAPTER

TWO

DEVELOPMENT

You can grab latest code on dev branch at Github.

Feel free to submit issues, pull requests are also welcome.

CHAPTER

THREE

TESTS

You can run testsuite this way:

```
python manage.py runtests.py
```

PYTHON MODULE INDEX

d