Django Tracking Analyzer Documentation

Release 0.2

José Luis Patiño Andrés

September 24, 2016

Contents

User actions tracking and analytics for Django sites

CHAPTER 1

Contents

1.1 Installation

1.1.1 Prerequisites

The next requirements will be installed together with Django Tracking Analyzer:

- Django +1.9 or later.
- Django Countries 3.4.1 or later.
- Django IPWare 1.1.5 or later.
- Django User Agents 0.3.0 or later.
- GeoIP2 2.3.0 or later.

1.1.2 Package installation

You can install the Django Tracking Analyzer package in the two traditional Python ways:

1. Automatically, by using PyPI (this is of course the preferred way):

```
pip install django-tracking-analyzer
```

2. Manually, by downloading the package and installing yourself:

```
wget https://github.com/jose-lpa/django-tracking-analyzer/archive/master.zip
unzip master.zip
cd django-tracking-analyzer-master/
python setup.py install
```

1.1.3 Configuration

Once the package is installed, the next two entries have to be added to the `INSTALLED_APPS` setting:

- `'django_user_agents'` (enables all the functionalities needed to get user agent data)
- `'tracking_analyzer'`

1.1.4 MaxMind® GeoLite2 datasets installation

In order to have IP geolocation and related data collection, DTA uses the GeoIP2 library, which should have been installed together with this package.

GeoIP2 uses MaxMind® geographical datasets to work, which location is pointed by the `GEOIP_PATH` Django setting. This means you have to dowload and install those datasets in the directory specified by that setting, which you can do in two different ways:

1. Automatically, by using the DTA management command `install_geoip_dataset`. This is the preferred and fastest way of doing it, and you should be ready to go by just running the command: `python setup.py install_geoip_dataset`.
2. Manually, by downloading yourself the datasets from the MaxMind® GeoLite2 site.

Once this last step is done, you should be ready to use Django Tracking Analyzer.

`install_geoip_dataset` management command

The management command `install_geoip_dataset` is available to help you download and install the MaxMind GeoLite2 datasets without any effort.

The only thing you have to make is to ensure the existence of the Django setting `GEOIP_PATH`. This setting specifies the directory where the GeoIP data files are located. It is unset by default, you have to set it to a directory where you want the datasets to be installed. A nice place is usually the root directory of your project code.

Once you have this setting, you can execute the management command and, if you don't have any GeoLite2 files in that directory, it will download and decompress the GeoLite2 "City" and "Country" datasets for you:

```
python manage.py install_geoip_dataset
```

In case you already have the datasets installed, the command will ask you if you really want to download another datasets and replace the existing ones with the latest downloaded version:

```
Seems that MaxMind dataset GeoLite2-Country.mmdb.gz is already installed in "GeoLite2-Country.mmdb.g
(y/n)
```

The process will run for both the "Country" and the "City" datasets.

Although the management command will everything you need in a simlpe run, you might want to specify some different download options, such the datasets files names or the download URL. You can do this via the next command parameters:

- `url`: Base URL where the MaxMind(R) datasets are available.
- `countries`: Remote file name for the MaxMind(R) Country dataset (compressed).
- `cities`: Remote file name for the MaxMind(R) City dataset (compressed).

1.2 Usage

1.2.1 Overview

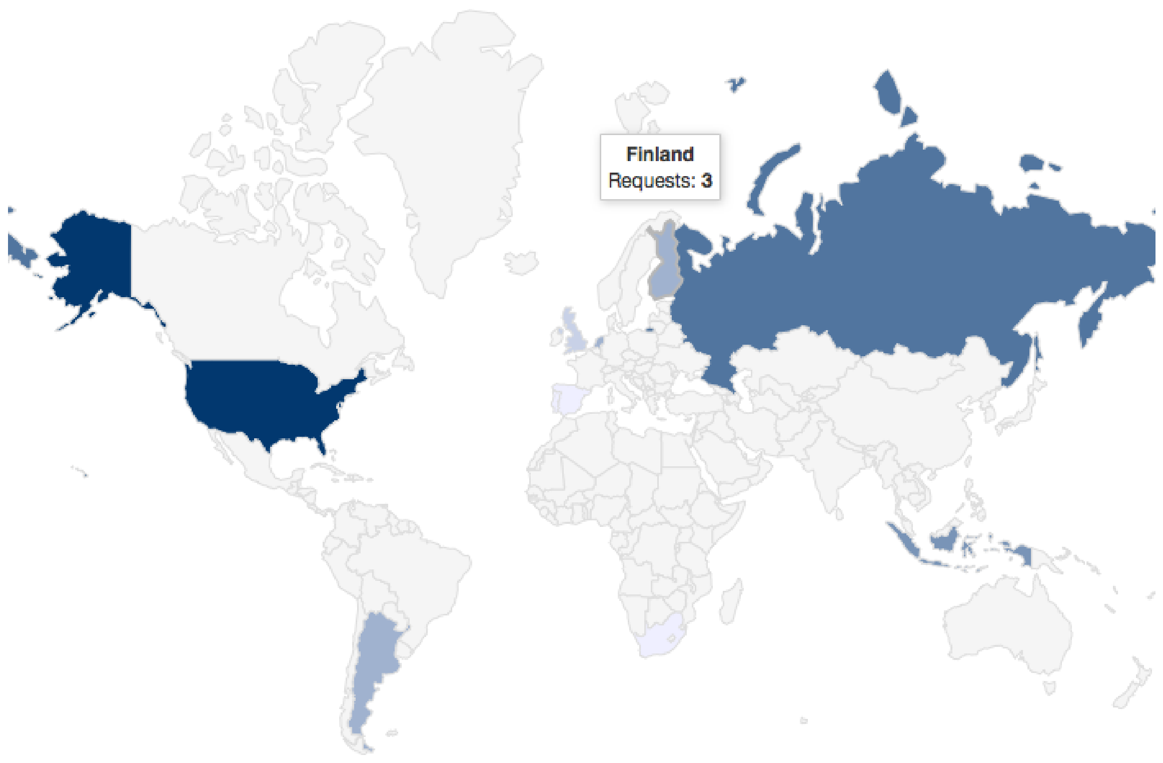
Django Tracking Analyzer is a Django application that aims to help you know in a simple and user-friendly way who the visitors of your site are, where they come from, what devices are they using to browse your site, what resources of your site they access, when and how many times.

In order to do this, DTA implements a database model `Tracker`, which will be created each time a user access certain resource, like a blog post, or performs certain action, like buying a product in your web shop.

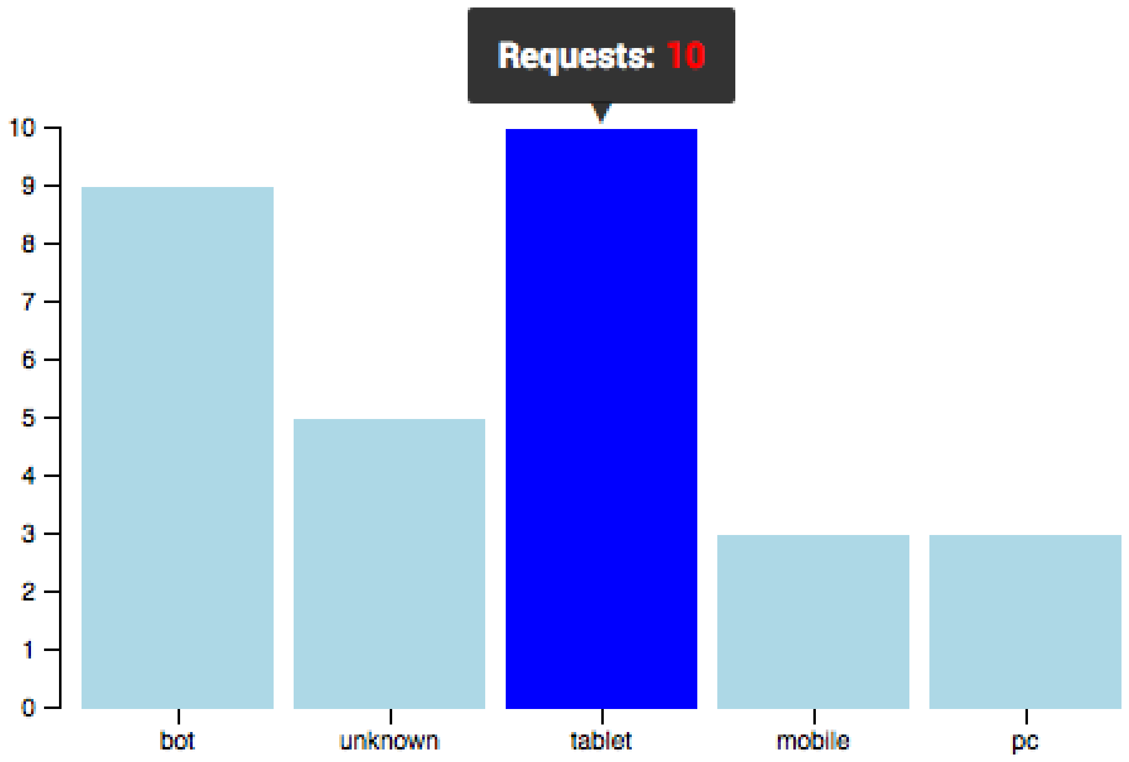
Then, using the Django admin interface, you can check the "Trackers" changelist in the "Django Tracking Analyzer" app admin, and you will see a changelist of all the user accesses with details about the requests, like the IP address, the country and city (if available), the device type, browser and system information.

And also, heading the traditional changelist page, you will be provided with some nice interactive graphics made in D3.js, to actually see all the data gathered in a visual fancy way:

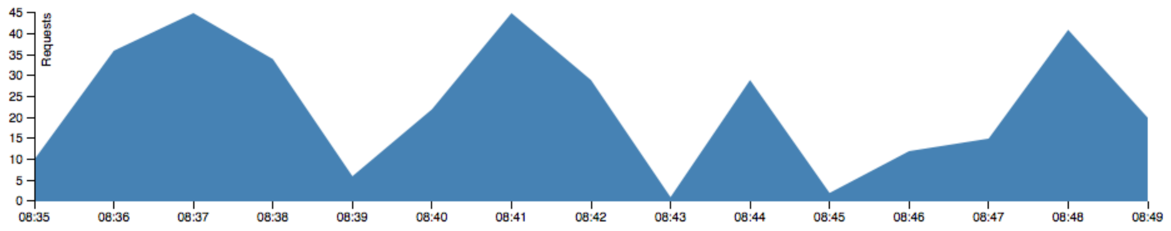
- A world map where you can see where the visitors of your site come from.



- A bars graph showing the number of requests per device type.



- A timeline showing the number of requests for the current changelist page.



1.2.2 Tracking requests or events

Collecting users data is as simple as relate a user request with a Django model that represents something you want to track: a blog post, an item in a webshop or anything else you have represented as a database model in your Django site.

The `Tracker` model then will relate a request with such an object of your choice, collecting all the relevant request data like geolocation, user device, browser and system version. When you have several `Tracker`'s collected, you can visualize the data and get an idea of what kind of users you site have or who is buying what products in your webshop.

A `Tracker` object can be created by using the special manager method `create_from_request`:

```
Tracker.objects.create_from_request(request, model_instance)
```

As you can see, the only things needed are an HTTP request object and a model instance. This makes it very easy to record tracks in any view of your project, which will be the place where a user interacts with some resource or performs some action: the Django views.

You can override the `get_object` method in a `DetailView`, if you are usign class-based views, or you can create the `Tracker` wherever you want in your function views. The only thing you need is a `django.http.HttpRequest` instance, which is of course available in any Django view, and a database model instance you want to track.

Examples

Let's see an example of how to track a user action. Imagine you have a web shop and you want to get a track of the users who buy products from it.

You could have a view in your code that, once a user submits a purchasing form, executes some code to perform the purchase/payment operations. In such a view, you can simply add the `Tracker` object creation after the purchase is completed, making a system record of the client.

```
from django.contrib.auth.decorators import login_required
from django.shortcuts import get_object_or_404, render_to_response

from tracking_analyzer.models import Tracker


@login_required
def purchase_item(request, product_pk):
    product = get_object_or_404(Product, pk=product_pk)

    # Here is your logic to perform the purchase...
    ...
    # ...and to make the payment and other needed operations.

    # Track the products purchases!
    Tracker.objects.create_from_request(self.request, product)

    return render_to_response(template, context_dict, RequestContext(request))
```

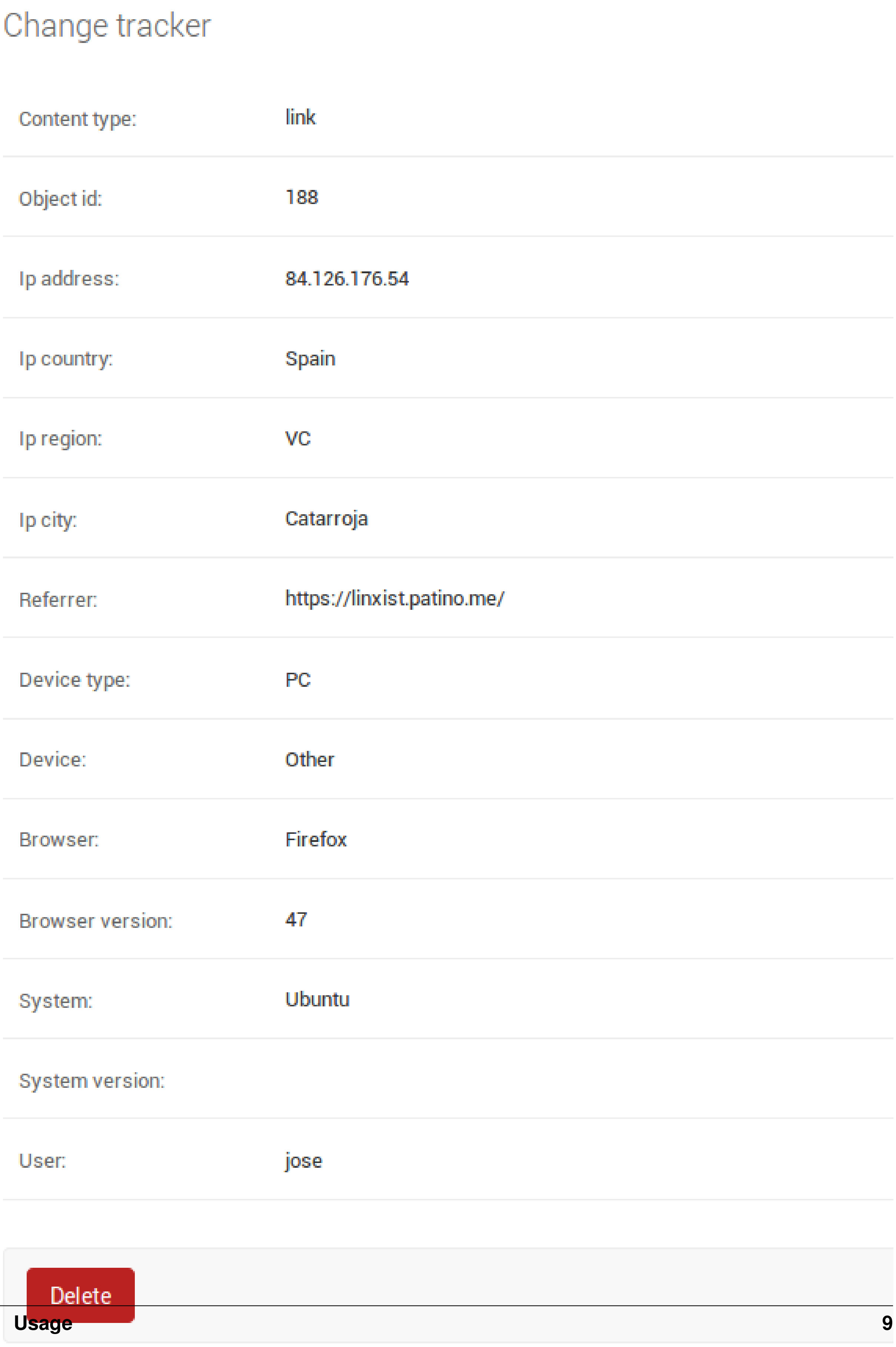
With that simple line added to your code, you'll have a detail log of every purchase request. Note that the special manager method `create_from_request` does everything for you in terms of collecting request information and relating it to the passed object instance.

In other environment, you might be using Django class-based views. Let's see an example in which you have a news site, so you have daily news posted in a main page and the users can read each notice by clicking the title. Then you might have a `DetailView` to show a notice by its slug, and so you can track the visitors to your news by using DTA in a similar way than before:

```
from django.views.generic import DetailView

from tracking_analyzer.models import Tracker

from news.models import NewsEntry


class NewsEntry(DetailView):
    model = NewsEntry

    def get_object(self, queryset=None):
        # Retrieve the news entry just using `get_object` functionality.
        entry = super(PostDetailView, self).get_object(queryset)

        # Track the users access to the news by entry!
        Tracker.objects.create_from_request(self.request, entry)
        return entry
```

As you can see, it's pretty simple and straightforward to start collecting user data from the requests to your site.

If you go to your Django admin, "Django Tracking Analyzer" - "Trackers" section, you will see a regular Django changelist of "trackers". Then if you click the "Details" link in one of them, you can see the request data overview as follows:



Change tracker

Content type:	link
Object id:	188
Ip address:	84.126.176.54
Ip country:	Spain
Ip region:	VC
Ip city:	Catarroja
Referrer:	https://linxist.patino.me/
Device type:	PC
Device:	Other
Browser:	Firefox
Browser version:	47
System:	Ubuntu
System version:	
User:	jose

1.3 Settings

Django Tracking Analyzer behaviour can be customized by using the next application settings in your project:

1.3.1 `GEOIP_PATH`

It defines the path where the GeoIP2 datasets are installed.

Although this is not a Django Tracking Analyzer setting, you have to set it up in order to get the GeoIP datasets working properly. Please check GeoIP2 Django documentation for detailed information.

- Default: **unset**

1.3.2 `TRACKING_ANALYZER_MAXMIND_URL`

Specifies the URL where the GeoIP datasets have to be retrieved from. It defaults to the MaxMind database official repository.

- Default: `'http://geolite.maxmind.com/download/geoip/database/'`

1.3.3 `TRACKING_ANALYZER_MAXMIND_COUNTRIES`

Specifies the file name for the *compressed* countries database.

- Default: `'GeoLite2-Country.mmdb.gz'`

1.3.4 `TRACKING_ANALYZER_MAXMIND_CITIES`

Specifies the file name for the *compressed* cities database.

- Default: `'GeoLite2-City.mmdb.gz'`

1.4 Contributing

If you want to add some functionality or you spot a bug and you want to fix it yourself, you can fork the Github repository and make all your changes in your local clone.

Once you are done, please submit all your changes by issuing a pull request against the `development` branch.

When you are writing your code, please take into account the following basic rules:

- Always include some unit tests for the new code you write or the bugs you fix. Or, update the existent unit tests, if necessary.
- Stick to PEP-8 styling.
- Make your pull requests to `development` branch.

1.4.1 Testing

We use Pytest to unit test Django Tracking Analyzer. You can run the tests in your local in 3 different ways, depending on what you want to check:

1. Simply run the unit tests. This should be enough for everybody:

   ```
   python setup.py test
   ```

2. Run tests with code coverage analysis:

   ```
   python setup.py test –pytest-args "–cov-report xml –cov tracking_analyzer tests/ –verbose –junit-xml=junit.xml
   –color=yes"
   ```

3. Run tests with coverage and Pylint/PEP8 checking:

   ```
   python setup.py test –pytest-args "–cov-report xml –cov tracking_analyzer tests/ –verbose –junit-xml=junit.xml
   –color=yes –pylint –pylint-rcfile=pylint.rc –pep8"
   ```

The last one will probably fail ;) but don't worry too much about that. Just make sure that at least the 1st testing command works well without any errors.

CHAPTER 2

License

Django Tracking Analyzer is licensed under the GNU GPLv3 license. This makes you able to modify and redistribute the code of this package freely, as well as to use it in your own Django sites or applications publicly.

The related software included in this package, such as D3.js or some code for the Javascript widgets, have its own licenses. In case you want to modify and use that software for your own purposes, please refer to their own licenses.

CHAPTER 3

Source code and contributions

The source code can be found on Github.

Bugs can be reported on the Github repository issues. Any collaboration will be very appreciated. Please see *Contributing* for more details.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search