Django Oscar Easyrec Documentation

Release 0.0.9

Jonathan Moss

February 07, 2015

Contents

Oscar is an open-source, domain driven ecommerce framework for Django and easyrec is a recommendation system. So naturally *django oscar easyrec* is a reusable app designed to integrate the two as seamlessly as possible.

Contents:

CHAPTER 1

Getting Started

1.1 Installation

From PyPI:

```
pip install django-oscar-easyrec
```

or from Github:

```
pip install git+git://github.com/tangentlabs/django-oscar-easyrec.git#egg=django-oscar-easyrec
```

Add `'easyrec'` to `INSTALLED_APPS`.

You will also need to install easyrec itself.

Instructions for installing Easyrec can be found on easyrec's sourceforge wiki

Note: If you want to make use of back tracking urls. Ensure you set the site url correctly in the tenants configuration section of `easyrec`.

I'd also heartily recommend reading through some of the basic concepts of `easyrec` to get a better idea of how to get the best out of it.

1.2 Configuration

Edit your `settings.py` to set the following settings:

```
EASYREC_ENDPOINT = 'http://127.0.0.1:8080/easyrec-web/'
EASYREC_TENANT_ID = '...'
EASYREC_API_KEY = '...'
```

In easyrec all items have an 'itemtype'. django-oscar-easyrec passes the product class name for this value. If the item type is not registered in easyrec it will send the default value of 'ITEM'.

So each of your product classes needs to manually added as an itemtype via easyrec's dashboard if you want them to be recorded separately.

Note: If you add itemtypes to easyrec you will need to restart your django project to ensure they are picked up correctly.

And that's it! All purchases, product views and reviews will automatically be pushed to easyrec.

CHAPTER 2

Getting Recommendations

django-oscar-easyrec comes with a templatetag allowing you to easily fetch recommendations and display them in your templates. There are a number supported template tags which do pretty much what they say:

```
{% load recommendations %}

{% user_recommendations request.user as recommendations %}
{% for recommended_product in recommendations %}
    <!-- Do your thing! -->
{% endfor %}

{% users_also_bought a_product request.user as recommendations %}
{% for recommended_product in recommendations %}
    <!-- Do your thing! -->
{% endfor %}

{% users_also_viewed a_product request.user as recommendations %}
{% for recommended_product in recommendations %}
    <!-- Do your thing! -->
{% endfor %}

{% products_rated_good product as recommendations %}
{% for recommended_product in recommendations %}
    <!-- Do your thing! -->
{% endfor %}

{% related_products product as recommendations %}
{% for recommended_product in recommendations %}
    <!-- Do your thing! -->
{% endfor %}
```

Each template tag provides a list of recommendations. Each recommendation is a dictionary containing a product object, and a tracking url. e.g.:

```
{
    "product": <Product>,
    "tracking_url": "http://somewhere.com"
}
```

If no recommendations are found then an empty list is returned. Each of these tags also supports a number of other optional parameters.

You can also call the recommendation functions directly:

```python
from easyrec.utils import get_gateway

easyrec = get_gateway()
recommendations = easyrec.get_user_recommendations(user.user_id)
recommendations = easyrec.get_other_users_also_bought(product.upc, user_id)
recommendations = easyrec.get_other_users_also_viewed(product.upc, user_id)
```

Recommendation Template Tags

3.1 user_recommendations

Returns a list of recommended items for a user

Syntax:

```
{% user_recommendations <user> [max_results 15] [requested_item_type "ITEM"] [action_type "VIEW"] %}
```

3.2 users_also_bought

Returns a list of recommended items based on users who bought this also bought X

Syntax:

```
{% users_also_bought <product> <user> [max_results 15] [requested_item_type "ITEM"] %}
```

3.3 users_also_viewed

Returns a list of recommended items based on users who viewed this also viewed X

Syntax:

```
{% users_also_viewed <product> <user> [max_results 15] [requested_item_type "ITEM"] %}
```

3.4 products_rated_good

Returns a list of recommended items based on users who rated this as good also rated X as good.

Syntax:

```
{% product_rated_good <product> <user> [max_results 15] [requested_item_type "ITEM"] %}
```

3.5 `related_products`

Returns a list of items related to the supplied one

Syntax:

```
{% related_products <product> <user> [max_results 15] [assoc_type "IS_RELATED"] [requested_item_type
```

3.6 Parameters

Permitted values for some of the optional parameters are more obvious than other so let go through them here just to make sure:

`max_results` The maximum number of products you want. The default is up to 15

`requested_item_type` The item type you want in the results. The default is "ITEM"

`assoc_type` The associate type between the products. The default is "IS_RELATED" but you have the following options: "BOUGHT_TOGETHER", "GOOD_RATED_TOGETHER", "IS_RELATED", "VIEWED_TOGETHER".

Getting Rankings

Also provided are a collection of template tags for getting community rankings of products. i.e. Their popularity within certain actions. Thing like most bought or best rated products. All these template tags share the same syntax:

```
{% <ranking_type> [time_range "ALL"] [max_results 15] [requested_item_type "ITEM"] %}
```

Each template tag provides a list of rankings in the same way as the recommendation tags. e.g. a dictionary containing a product object, and a tracking url.:

```
{
    "product": <Product>,
    "tracking_url": "http://somewhere.com"
}
```

If no recommendations are found then an empty list is returned.

CHAPTER 5

Rankings Template Tags

5.1 most_viewed

Returns the most viewed products

5.2 most_bought

Provides the most purchased items

5.3 most_rated

Products with the most rating. This is not the best rated products, but those that have the greatest number of ratings in total, high or low.

5.4 best_rated

The products with the best ratings.

5.5 worst_rated

The products with the worst rated over all.

Example:

```
{% load rankings %}

{% most_viewed as rankings %}
<ol>
{% for item in rankings %}
    <li>
        <a href="{{ item.tracking_url }}">
            {{ item.product.title }}
        </a>
    </li>
```

```
{% endfor %}
</ol>
```

5.6 Parameters

Permitted values for some of the optional parameters or more obvious than other so here they are explained:

`time_range` The range over which you want the ranking. Options include: “DAY”, “WEEK”, “MONTH”, “ALL”. The default is “ALL”

`max_results` The maximum number of products you want. The default is up to 15

`requested_item_type` The item type you want in the results. The default is “ITEM”

CHAPTER 6

Asynchronous Actions

As of *django-oscar-easyrec* 0.6 it is now possible to have collected actions (views, ratings and purchases) delivered to easyrec asynchronously with the aid of the most excellent Celery. Installing and running celery is beyond the scope of this documentation but can be found in the Celery install guide

Once you have Celery up and running all that is needed to get *django-oscar-haystack* using it to deliver actions asynchronously is to add the following to your settings file:

```
EASYREC_ASYNC = True
```

CHAPTER 7

Settings

`EASYREC_ENDPOINT`

A url pointing to the location of your easyrec install.

Syntax:

```
EASYREC_ENDPOINT = 'http://127.0.0.1:9090/easyrec-web'
```

`EASYREC_TENANT_ID`

This is effectively the user name for your *easyrec* account.

Syntax:

```
EASYREC_TENANT_ID = 'EASYREC_DEMO'
```

`EASYREC_API_KEY`

Effectively the password for you *easyrec* account.

Syntax:

```
EASYREC_API_KEY = '8ab9dc3ffcdac576d0f298043a60517a'
```

`EASYREC_ASYNC`

Alows you to switch the sending of actions to *easyrec* asynchronously. This requires Celery to be installed any running. Defaults to *False*

Syntax:

```
EASYREC_ASYNC = False
```

CHAPTER 8

Indices and tables

- *search*