Trigger Happy Documentation

Release 0.13.3

foxmask

Jul 08, 2017

Contents

CHAPTER 1

Contents:

As it exists tools to make his blog or website or even his own cloud system, “Trigger Happy” is a free software that provides a bridge to automatically share data between popular services you use on the web. And instead of giving your credentials to them, keep them with your own **Trigger Happy** to keep the control of your data !

Quickstart

We will say we start from scratch. Assuming you already have python3.6 installed, with redis too.

Create a virtualenv

We just create a virtualenv with python 3.6 (or 3.5)

```
python3.6 -m venv myproject
cd $_
source bin/activate
```

Install from GitHub

We install Trigger-Happy from Pypi

```
git clone https://github.com/foxmask/django-th.git
cd django-th
pip install -e .[min]
```

Database

```
python manage.py migrate
python manage.py createsuperuser
```

Start the application

```
python manage.py runserver &
```

Now open your browser and go to http://127.0.0.1:8000/th/ to start using the application

Adding the service Wallabag from the Admin

Admin Home of Trigger Happy :

click add from

and fill the fields.

For the service RSS (dont check auth required) and Wallabag (check auth required)

This will give something like :

Activating the service

Now that the 2 service RSS and Wallabag are enabled, go activate them for you :

“Activated services” (http://127.0.0.1:8000/th/service/):

Why this process from admin and non admin part ?

- The project is hosted by yourself for your own need, but the project is able to handle trigger for your and your friends if you need.
- Thus the ‘admin’ who hosts the project need to do some work of his admin part to add the service he will offer to user
- Thus the user will go the his “my activated services” page to activate his service too.
- But as you are all alone for the moment, you have the two hats : admin and end user, this is why you will need to do the two steps “Adding the service wallabag from the Admin” and “Activating the service”

Create a trigger

Once all of this is done, go back to the main page http://127.0.0.1:8000/th/ and create your first trigger

Installation

TriggerHappy can be installed inside an existing project, or from scratch

Installation from scratch

We just create a virtualenv with python 3.6 (or 3.5)

```
python3.6 -m venv myproject
cd $_
source bin/activate
```

then you can continue with one of the two choice “From GitHub” or “From Pypi”

Installation from an existing project

```
cd /to/the/path/of/my/existing/project
source bin/activate   # (if you have a virtualenv)
```

then you can continue with one of the two choice “From GitHub” or “From Pypi”

Installation From GitHub

```
git clone https://github.com/foxmask/django-th.git
```

then continue by installing :

```
cd django-th
pip install -e .[min]
```

Installation From Pypi

```
pip install django-th[all]
```

or to make your own “recipe”, for example to install some of the component and not all of them:

```
pip install django-th[min]   # will just install rss and Wallabag
pip install django-th[rss,wallabag]
pip install django-th[rss,twitter,wallabag,github]
```

Once it’s done, you can continue to the [configuration process](http://trigger-happy.readthedocs.org/en/latest/configuration.html)

Requirements

- Python 3.5.x or 3.6.x
- Redis
- DjangoRestFramework
- Django
- Arrow
- Django-formtools
- Django-js-reverse

- Django-Redis
- Pypandoc
- Requests-oAuthlib

for evernote support

- Evernote for python 3
- libtidy-dev

The latest libtidy-dev should be installed with your operating system package manager, not from pip.

On a Debian/Ubuntu system:

```
apt-get install libtidy-dev
```

for github support

- github

for pelican support

- awesome-slugify

for pocket support

- pocket

for pushbullet support

- pushbullet.py

for redis support

- django-redis

for rss support

- feedparser

for taiga support

- python-taiga

for slack support

- requests

for todoist support

- todoist-python

for trello support

- trello
- pypandoc

Pandoc is also needed of the system, that you can install on a Debian/Ubuntu system like this:

```
apt-get install pandoc
```

for twitter support

- twython

for wallabag support

- wallabag_api

Configuration

Here are the details that will permit to make working the application correctly

setup urls.py

If TriggerHappy is your only one project installed in your virtualenv, go to "setup settings.py" this setup is just needed only when you add TriggerHappy to an **existing** application

add this line to the urls.py to be able to use the complete application

```
url(r'', include('django_th.urls')),
```

this will give something like

```
from django.conf.urls import patterns, include, url
from django.contrib import admin

urlpatterns = patterns('',
    # Examples:
    # url(r'^$', 'th.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),
    url(r'', include('django_th.urls')),
)
```

setup settings.py

add the module django_th, and its friends, to the INSTALLED_APPS

```
INSTALLED_APPS = (
     ...
     'formtools',
     'django_js_reverse',
     'rest_framework',
     'django_th',
     'th_rss',
     # uncomment the lines to enable the service you need
     # 'th_evernote',
     # 'th_github',
     # 'th_instapush',
     # 'th_pelican',
     # 'th_pocket',
     # 'th_pushbullet',
     # 'th_todoist',
     # 'th_trello',
     # 'th_twitter',
     'th_wallabag',
 )
```

do not forget to uncomment one of the service th_pocket, th_evernote (and then evernote also) th_twitter, th_trello, th_github otherwise, the application wont work.

setup for testing/debugging purpose

```
DEBUG = True
ALLOWED_HOSTS = ['*']
```

setup for production purpose

```
DEBUG = False
ALLOWED_HOSTS = ['127.0.0.1', 'localhost']
```

or set the hostname of your own domain

```
DEBUG = False
ALLOWED_HOSTS = ['mydomain.com']
```

setup th_settings.py

in the th_settings.py file, setup the TH_SERVICES

TH_SERVICES

TH_SERVICES is a list of the services, like for example,

```
TH_SERVICES = (
    # uncomment the lines to enable the service you need
    # uncomment the lines to enable the service you need
    # 'th_evernote.my_evernote.ServiceEvernote',
    # 'th_github.my_github.ServiceGithub',
    # 'th_instapush.my_instapush.ServiceInstapush',
    # 'th_pelican.my_pelican.ServicePelican',
    # 'th_pocket.my_pocket.ServicePocket',
    # 'th_pushbullet.my_pushbullet.ServicePushbullet',
    'th_rss.my_rss.ServiceRss',
    # 'th_todoist.my_todoist.ServiceTodoist',
    # 'th_trello.my_trello.ServiceTrello',
    # 'th_twitter.my_twitter.ServiceTwitter',
    'th_wallabag.my_wallabag.ServiceWallabag',
)
```

do not forget to uncomment one of the line to enable another service, or the application wont work.

Cache

They are necessary if you want to be able to follow the log and set the cache

For each TriggerHappy component, define one cache like below

```
CACHES = {
    'default':
    {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': BASE_DIR + '/cache/',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 1000
        }
    },
    # Evernote Cache
    'th_evernote':
    {
        'TIMEOUT': 500,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/1",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
    # GitHub
    'th_github':
    {
        'TIMEOUT': 3600,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/2",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
    # Pelican
    'th_pelican':
    {
        'TIMEOUT': 3600,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/3",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
    # Pocket Cache
    'th_pocket':
    {
        'TIMEOUT': 500,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/4",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
    # Pushbullet
    'th_pushbullet':
    {
        'TIMEOUT': 3600,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/5",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
```

```
        }
    },
    # RSS Cache
    'th_rss':
    {
        'TIMEOUT': 500,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/6",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
    # Todoist
    'th_todoist':
    {
        'TIMEOUT': 3600,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/7",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
    # Trello
    'th_trello':
    {
        'TIMEOUT': 3600,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/8",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
    # Twitter Cache
    'th_twitter':
    {
        'TIMEOUT': 500,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/9",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
    # Wallabag
    'th_wallabag':
    {
        'TIMEOUT': 3600,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/10",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
    'redis-cache':
    {
        'TIMEOUT': 3600,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://localhost:6379/11",
        "OPTIONS": {
```

```
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
            "MAX_ENTRIES": 5000,
        }
    },
    'django_th':
    {
        'TIMEOUT': 3600,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://localhost:6379/12",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
            "MAX_ENTRIES": 5000,
        }
    },
}
```

in the settings, 'default' may already exist in your settings.py, so don't use it, otherwise, if it doesn't, django will complain, so add it.

Logging

in the LOGGING add to loggers

```
LOGGING = {
    'handlers': {
        ...
        'file': {
            'level': 'INFO',
            'class': 'logging.handlers.RotatingFileHandler',
            'filename': BASE_DIR + '/trigger_happy.log',
            'maxBytes': 61280,
            'backupCount': 3,
            'formatter': 'verbose',

        },
    }
    'loggers':
    {
        ...
        'django_th.trigger_happy': {
            'handlers': ['console', 'file'],
            'level': 'INFO',
        }
    }
}
```

Once this is done we can create tasks in the crontab :

Suppose my virtualenv is created in /home/trigger-happy and the django app is located in /home/trigger-happy/th :

```
*/12 * * * * . /home/trigger-happy/bin/activate && cd /home/trigger-happy/th/ && ./
↪manage.py read
*/15 * * * * . /home/trigger-happy/bin/activate && cd /home/trigger-happy/th/ && ./
↪manage.py publish
*/20 * * * * . /home/trigger-happy/bin/activate && cd /home/trigger-happy/th/ && ./
↪manage.py recycle
```

Running

SETUP THE DATABASE AND RUNNING THE APPLICAITON

Update the database

Once the settings step is done, enter the following command to sync the database :

```
python manage.py migrate
```

If you meet some errors with this command, have a look at MIGRATION_0.10.x_to_0.11.x.rst file

If you are installing the project from scratch, do not forget to create a super user:

```
python manage.py createsuperuser
```

Start the application

```
python manage.py runserver
```

Now open your browser and go to 127.0.0.1:8000/th/ to start using the application

Usage

Activating services :

The user activates the service for their own need. If the service requires an external authentication, he will be redirected to the service which will ask him the authorization to acces the user's account. Once it's done, goes back to django-trigger-happy to finish and record the "auth token".

Using the activated services :

a set of 3 pages will ask to the user information that will permit to trigger data from a service "provider" to a service "consumer".

For example :

- page 1 : the user gives a RSS feed
- page 2 : the user gives the name of the notebook where notes will be stored and a tag if he wants
- page 3 : the user gives a description

Fire the Triggers :

Grabbing data and publishing data are done each 12min and 15min from your crontab

Services

This page cover the services that are handled by TriggerHappy, and will guide you through their installation

Common Process

For all the services, the installation is the same :

- modifications of settings.py
- creation of the table of the services (if needed)
- from the admin panel, activation of the service (if needed)

Activate the services

to activate a service, you will need to follow those steps

- Requesting a key to the Services
- Adding the key to your settings file
- Adding the service from the Admin
- Activating the service from your account from the public part of the website
- Why this process ?

in details this gives us :

Requesting a key to the Services

For each service, Trigger Happy expects to have some consumer key coming from the wanted service. So for each service, you need to register an account on each of this service, then required a key.

You can have a look at the README of Twitter, or README of Pocket

Adding the key to the th_settings

Once you own the keys., You add them to the th_settings.py file in

```
TH_<SERVICE_NAME> = (
    'consumer_key' => 'foobar',
    'consumer_token' => 'blabla'
)
```

For example for Twitter :

```
TH_TWITTER = {
    'consumer_key': 'abcdefghijklmnopqrstuvwxyz',
    'consumer_secret': 'abcdefghijklmnopqrstuvwxyz',
}
```

IMPORTANT :

With all the service you will enable, to avoid to share your key by accident, It's strongly recommended that you put all of them in a separate local_settings.py that you include at the end of the main settings.py

Adding the service from the Admin

Once you did **python manage.py migrate** and followed the standard process to bootstrap the application, go to the admin panel of the application.

Admin Home of Trigger Happy :

Admin list of activated services if Trigger Happy :

Admin Detail of one service of Trigger Happy :

Activating the service from your account from the public part of the website

Once your services are setup from the admin, you can go on the public part of the website and activate the service you need.

"My activated services" :

Why this process ?

- it is simple : actually, to use Trigger Happy you need to install and host it by yourself, and so, you need to "declare" for each service your instance of TriggerHappy to the service provider.
- Other details : you need to activate the service from the admin panel, BECAUSE, TriggerHappy is planed to be used by many other users soon. So the admin of the instance of TriggerHappy will decide if he wants to offer the possibility to use this service of this other one. Once the admin has done his job, the end user, from the "public part" can go to the list of services and add the new one etc.

Supported services

Here are the service that will follow almost the same previous path

Evernote

Service Description:

This service permits to take notes, photos, schedules things and so on

modifications of settings.py

1. INSTALLED_APPS :

add or uncomment the following lines

```
INSTALLED_APPS = (
    # 'evernote',
    # 'th_evernote',
)
```

to get

```
INSTALLED_APPS = (
    'evernote',
    'th_evernote',
)
```

2. Cache :

After the default cache add :

```
CACHES = {
    'default':
    {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': BASE_DIR + '/cache/',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 1000
        }
    },
    # Evernote Cache
    'th_evernote':
    {
        'TIMEOUT': 500,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/1",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
```

modifications of th_settings.py

1. TH_SERVICES

add or uncomment the following line

```
TH_SERVICES = (
    # 'th_evernote.my_evernote.ServiceEvernote',
)
```

to get

```
TH_SERVICES = (
    'th_evernote.my_evernote.ServiceEvernote',
)
```

2. The service keys

It's strongly recommended that your put the following in a local_settings.py, to avoid to accidentally push this to a public repository

```
TH_EVERNOTE = {
    'sandbox': True,  # set it to False when in production
    'consumer_key': 'my key',
    'consumer_secret': 'my secret',
}
```

creation of the table of the services

enter the following command

```
python manage.py migrate
```

from the admin panel, activation of the service

from http://yourdomain.com/admin/django_th/servicesactivated/add/

- Select “Evernote”,
- Set the Status to “Enabled”
- Check Auth Required: this will permit to redirect the user (or you) to Evernote website to confirm the access of the Evernote account
- Fill a description

GitHub

Service Description:

Powerful collaboration, code review, and code management for open source and private projects. Public projects are always free.

modifications of settings.py

1. INSTALLED_APPS :

add or uncomment the following line

```
INSTALLED_APPS = (
    # 'th_github',
)
```

to get

```
INSTALLED_APPS = (
    'th_github',
)
```

2. Cache :

After the default cache add :

```
CACHES = {
    'default':
    {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': BASE_DIR + '/cache/',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 1000
        }
```

```
    },
    # GitHub
    'th_github':
    {
        'TIMEOUT': 3600,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/7",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
```

modifications of th_settings.py

1. TH_SERVICES

add or uncomment the following line

```
TH_SERVICES = (
    # 'th_github.my_github.ServiceGithub',
)
```

to get

```
TH_SERVICES = (
    'th_github.my_github.ServiceGithub',
)
```

2. The service keys

It's strongly recommended that your put the following in a local_settings.py, to avoid to accidentally push this to a public repository

```
TH_GITHUB = {
    'username': 'username',
    'password': 'password',
    'consumer_key': 'my key',
    'consumer_secret': 'my secret'
}
```

creation of the table of the services

enter the following command

```
python manage.py migrate
```

from the admin panel, activation of the service

from http://yourdomain.com/admin/django_th/servicesactivated/add/

- Select "GitHub",
- Set the Status to "Enabled"

- Check Auth Required: this will permit to redirect the user (or you) to GitHub website to confirm the access of the GitHub account
- Fill a description

Instapush

Service Description:

Notification service

modifications of settings.py

1. INSTALLED_APPS :

uncomment the following line

```
INSTALLED_APPS = (
    # 'th_instapsuh',
)
```

to get

```
INSTALLED_APPS = (
    'th_instapsuh',
)
```

modifications of th_settings.py

1. TH_SERVICES

uncomment the following line

```
TH_SERVICES = (
    # 'th_instapush.my_instapush.ServiceInstapush',
)
```

to get

```
TH_SERVICES = (
    'th_instapush.my_instapush.ServiceInstapush',
)
```

creation of the table of the services

enter the following command

```
python manage.py migrate
```

from the admin panel, activation of the service

from http://yourdomain.com/admin/django_th/servicesactivated/add/

- Select “Instapush”,
- Set the Status to “Enabled”
- Check Auth Required: do not check it
- Fill a description

Pelican

Service Description:

Pelican Static Site Generator

modifications of settings.py

1. INSTALLED_APPS :

add or uncomment the following line

```
INSTALLED_APPS = (
    # 'th_pelican',
)
```

to get

```
INSTALLED_APPS = (
    'th_pelican',
)
```

2. Cache :

After the default cache add :

```
CACHES = {
    'default':
    {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': BASE_DIR + '/cache/',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 1000
        }
    },
    # Pelican
    'th_pelican':
    {
        'TIMEOUT': 3600,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/8",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
```

```
        }
    },
```

modifications of th_settings.py

1. TH_SERVICES

add or uncomment the following line

```
TH_SERVICES = (
    # 'th_pelican.my_pelican.ServicePelican',
)
```

to get

```
TH_SERVICES = (
    'th_pelican.my_pelican.ServicePelican',
)
```

4. Pelican Author :

Set an author that will be added to the creation of each post

```
TH_PELICAN_AUTHOR = 'Foxmask'
```

creation of the table of the services

enter the following command

```
python manage.py migrate
```

from the admin panel, activation of the service

from http://yourdomain.com/admin/django_th/servicesactivated/add/

- Select “Pelican”,
- Set the Status to “Enabled”
- Uncheck “Auth Required”: this service does not required an authorization to access to something
- Fill a description

Pocket

Service Description:

a “Read it Later” service

modifications of settings.py

1. INSTALLED_APPS :

add or uncomment the following line

```
INSTALLED_APPS = (
    # 'th_pocket',
)
```

to get

```
INSTALLED_APPS = (
    'th_pocket',
)
```

2. Cache :

After the default cache add :

```
CACHES = {
    'default':
    {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': BASE_DIR + '/cache/',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 1000
        }
    },
    # Pocket Cache
    'th_pocket':
    {
        'TIMEOUT': 500,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/5",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
```

modifications of th_settings.py

1. TH_SERVICES

uncomment the following line

```
TH_SERVICES = (
    # 'th_pocket.my_pocket.ServicePocket',
)
```

to get

```
TH_SERVICES = (
    'th_pocket.my_pocket.ServicePocket',
)
```

2. The service keys

It's strongly recommended that your put the following in a local_settings.py, to avoid to accidentally push this to a public repository

```
TH_POCKET = {
    # get your credential by subscribing to http://getpocket.com/developer/
    'consumer_key': '<your pocket key>',
}
```

creation of the table of the services

enter the following command

```
python manage.py migrate
```

from the admin panel, activation of the service

from http://yourdomain.com/admin/django_th/servicesactivated/add/

- Select "Pocket",
- Set the Status to "Enabled"
- Check Auth Required: this will permit to redirect the user (or you) to Pocket website to confirm the access of the Pocket account
- Fill a description

Pushbullet

Service Description:

Your devices working better together

Nota : to be able to work, this service requires that your host uses HTTPS

modifications of settings.py

1. INSTALLED_APPS :

uncomment the following line

```
INSTALLED_APPS = (
    # 'th_pushbullet',
)
```

to get

```
INSTALLED_APPS = (
    'th_pushbullet',
)
```

2. Cache :

After the default cache add :

```
CACHES = {
    'default':
    {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': BASE_DIR + '/cache/',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 1000
        }
    },
    # Pushbullet Cache
    'th_pushbullet':
    {
        'TIMEOUT': 500,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/12",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
```

modifications of th_settings.py

1. TH_SERVICES

add or uncomment the following line

```
TH_SERVICES = (
    # 'th_pushbullet.my_pushbullet.ServicePushbullet',
)
```

to get

```
TH_SERVICES = (
    'th_pushbullet.my_pushbullet.ServicePushbullet',
)
```

2. The service keys

It's strongly recommended that your put the following in a local_settings.py, to avoid to accidentally push this to a public repository

```
TH_PUSHBULLET = {
    # get your credential by subscribing to
    # https://www.pushbullet.com/#settings/clients
    'client_id': '<your pushbulet id>',
    'client_secret': '<your pushbulet secret>',
}
```

creation of the table of the services

enter the following command

```
python manage.py migrate
```

from the admin panel, activation of the service

from http://yourdomain.com/admin/django_th/servicesactivated/add/

- Select “Pushbullet”,
- Set the Status to “Enabled”
- Check Auth Required: this will permit to redirect to the user (or you) to Pushbullet to ask to confirm the access to his/your Pushbullet account
- Fill a description

RSS

Service Description:

Service that grab RSS all around the web or create also RSS from other services

modifications of settings.py

1. INSTALLED_APPS :

add or uncomment the following line

```
INSTALLED_APPS = (
    # 'th_rss',
)
```

to get

```
INSTALLED_APPS = (
    'th_rss',
)
```

2. Cache :

After the default cache add :

```
CACHES = {
    'default':
    {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': BASE_DIR + '/cache/',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 1000
        }
    },
    # RSS Cache
    'th_rss':
    {
        'TIMEOUT': 500,
```

```
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/5",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
```

modifications of th_settings.py

1. TH_SERVICES

uncomment the following line

```
TH_SERVICES = (
    # 'th_rss.my_rss.ServiceRss',
)
```

to get

```
TH_SERVICES = (
    'th_rss.my_rss.ServiceRss',
)
```

creation of the table of the services

enter the following command

```
python manage.py migrate
```

from the admin panel, activation of the service

from http://yourdomain.com/admin/django_th/servicesactivated/add/

- Select “RSS”,
- Set the Status to “Enabled”
- Uncheck “Auth Required”: this service does not required an authorization to access to something
- Fill a description

Slack

Service Description:

A messaging app for teams who put robots on Mars

modifications of settings.py

1. INSTALLED_APPS :

add or uncomment the following line

```
INSTALLED_APPS = (
    # 'th_slack',
)
```

to get

```
INSTALLED_APPS = (
    'th_slack',
)
```

modifications of th_settings.py

1. TH_SERVICES

add or uncomment the following line

```
TH_SERVICES = (
    # 'th_slack.my_slack.ServiceSlack',
)
```

to get

```
TH_SERVICES = (
    'th_slack.my_slack.ServiceSlack',
)
```

creation of the table of the services

enter the following command

```
python manage.py migrate
```

from the admin panel, activation of the service

from http://yourdomain.com/admin/django_th/servicesactivated/add/

- Select “Slack”,
- Set the Status to “Enabled”
- Fill a description

Taiga

Service Description:

Taiga is a project management platform for agile developers & designers and project managers who want a beautiful tool that makes work truly enjoyable.

modifications of settings.py

1. INSTALLED_APPS :

add or uncomment the following line

```
INSTALLED_APPS = (
    # 'th_taiga',
)
```

to get

```
INSTALLED_APPS = (
    'th_taiga',
)
```

modifications of th_settings.py

1. TH_SERVICES

add or uncomment the following line

```
TH_SERVICES = (
    # 'th_taiga.my_taiga.ServiceTaiga',
)
```

to get

```
TH_SERVICES = (
    'th_taiga.my_taiga.ServiceTaiga',
)
```

creation of the table of the services

enter the following command

```
python manage.py migrate
```

from the admin panel, activation of the service

from http://yourdomain.com/admin/django_th/servicesactivated/add/

- Select “Taiga”,
- Set the Status to “Enabled”
- Fill a description

Todoist

Service Description:

a Tasks Managements

Nota : to be able to work, this service requires that your host uses HTTPS

modifications of settings.py

1. INSTALLED_APPS :

add or uncomment the following line

```
INSTALLED_APPS = (
    # 'th_todoist',
)
```

to get

```
INSTALLED_APPS = (
    'th_todoist',
)
```

2. Cache :

After the default cache add :

```
CACHES = {
    'default':
    {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': BASE_DIR + '/cache/',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 1000
        }
    },
    # Todoist Cache
    'th_todoist':
    {
        'TIMEOUT': 500,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/11",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
```

modifications of th_settings.py

1. TH_SERVICES

add or uncomment the following line

```
TH_SERVICES = (
    # 'th_todoist.my_todoist.ServiceTodoist',
)
```

to get

```
TH_SERVICES = (
    'th_todoist.my_todoist.ServiceTodoist',
)
```

2. The service keys

It's strongly recommended that your put the following in a local_settings.py, to avoid to accidentally push this to a public repository

```
TH_TODOIST = {
    # get your credential by subscribing to
    # https://developer.todoist.com/appconsole.html
    'client_id': '<your todoist id>',
    'client_secret': '<your todoist secret>',
}
```

creation of the table of the services

enter the following command

```
python manage.py migrate
```

from the admin panel, activation of the service

from http://yourdomain.com/admin/django_th/servicesactivated/add/

- Select "Todoist",
- Set the Status to "Enabled"
- Check Auth Required: this will permit to redirect the user (or you) to Todoist website to confirm the access of the Todoist account
- Fill a description

Trello

Service Description:

a Kanban application

modifications of settings.py

1. INSTALLED_APPS :

add or uncomment the following line

```
INSTALLED_APPS = (
    # 'th_trello',
)
```

to get

```
INSTALLED_APPS = (
    'th_trello',
)
```

2. Cache :

After the default cache add :

```
CACHES = {
    'default':
    {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': BASE_DIR + '/cache/',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 1000
        }
    },
    # Trello Cache
    'th_trello':
    {
        'TIMEOUT': 500,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/5",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
```

modifications of th_settings.py

1. TH_SERVICES

add or uncomment the following line

```
TH_SERVICES = (
    # 'th_trello.my_trello.ServiceTrello',
)
```

to get

```
TH_SERVICES = (
    'th_trello.my_trello.ServiceTrello',
)
```

2. The service keys

It's strongly recommended that your put the following in a local_settings.py, to avoid to accidentally push this to a public repository

```
TH_TRELLO = {
    'consumer_key': '<your trello key>',
    'consumer_secret': '<your trello secret>',
}
```

creation of the table of the services

enter the following command

```
python manage.py migrate
```

from the admin panel, activation of the service

from http://yourdomain.com/admin/django_th/servicesactivated/add/

- Select “Trello”,
- Set the Status to “Enabled”
- Check Auth Required: this will permit to redirect the user (or you) to Trello website to confirm the access of the Trello account
- Fill a description

Twitter

Service Description:

a Social Network

modifications of settings.py

1. INSTALLED_APPS :

add or uncomment the following line

```
INSTALLED_APPS = (
    # 'th_twitter',
)
```

to get

```
INSTALLED_APPS = (
    'th_twitter',
)
```

2. Cache :

After the default cache add :

```
CACHES = {
    'default':
    {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': BASE_DIR + '/cache/',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 1000
        }
    },
```

```
    # Twitter Cache
    'th_twitter':
    {
        'TIMEOUT': 500,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/5",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
```

modifications of th_settings.py

1. TH_SERVICES

add or uncomment the following line

```
TH_SERVICES = (
    # 'th_twitter.my_twitter.ServiceTwitter',
)
```

to get

```
TH_SERVICES = (
    'th_twitter.my_twitter.ServiceTwitter',
)
```

2. The service keys

It's strongly recommended that your put the following in a local_settings.py, to avoid to accidentally push this to a public repository

```
TH_TWITTER = {
    # get your credential by subscribing to
    # https://dev.twitter.com/
    'consumer_key': '<your twitter key>',
    'consumer_secret': '<your twitter secret>',
}
```

creation of the table of the services

enter the following command

```
python manage.py migrate
```

from the admin panel, activation of the service

from http://yourdomain.com/admin/django_th/servicesactivated/add/

- Select "Twitter",
- Set the Status to "Enabled"

- Check Auth Required: this will permit to redirect the user (or you) to Twitter website to confirm the access of the Twitter account
- Fill a description

Wallabag

Service Description:

a self hostable application for saving web pages

modifications of settings.py

add or uncomment the following lines

```
INSTALLED_APPS = (
    # 'th_wallabag',
)
```

to get

1. INSTALLED_APPS :

```
INSTALLED_APPS = (
    'th_wallabag',
)
```

2. Cache :

After the default cache add :

```
CACHES = {
    'default':
    {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': BASE_DIR + '/cache/',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 1000
        }
    },
    # Wallabag Cache
    'th_wallabag':
    {
        'TIMEOUT': 500,
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/9",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
```

modifications of th_settings.py

1. TH_SERVICES

add or uncomment the following line

```
TH_SERVICES = (
    # 'th_wallabag.my_wallabag.ServiceWallabag',
)
```

to get

```
TH_SERVICES = (
    'th_wallabag.my_wallabag.ServiceWallabag',
)
```

4. The service keys

Those will be required to be filled when activating the service for each user

Have a look at https://github.com/foxmask/wallabag_api/blob/master/README.rst for more details about them

creation of the table of the services

enter the following command

```
python manage.py migrate
```

from the admin panel : activation of the service

from http://yourdomain.com/admin/django_th/servicesactivated/add/

- Select "Wallabag",
- Set the Status to "Enabled"
- Check Auth Required: this will permit to redirect the user (or you) to your Wallabag application which will request a token
- Check Self Hosted: this will permit to enter the details about the service key we speak from point 4
- Fill a description

from "My Activated Service" page

Now go to the page of "My Activated services" to enable it http://yourdomain.com/th/service/ by pressing the blue button "Activate a new service"

then fill the fields that are required with the parameters, you got from point 4 earlier

Create a new module

2 ways to reach the goal to "bootstrap" a new TriggerHappy module :

1 - django-th-ansible :

just simple and fast ;)

with git, clone [django-th-ansible](https://github.com/foxmask/django-th-ansible), modify the site.yml file and run:

```
ansible-playbook -i site.yml
```

Now aour new module is ready to be customized for your new service (template, models and so on).

2 - django-th-dummy :

Introduction :

You can start a new module by cloning the project Django Th Dummy which is a vanilla django module, ready to be used, after you've replaced the name of the form/model/class we'll see below

Once you've cloned it, rename the folder th_dummy to the name of your choice.

Below we'll keep the name dummy to continue our explanation

Forms :

the form **th_dummy/forms.py** provides 3 forms :

- **DummyForm** a modelForm
- **DummyFormProvider** which extends DummyForm
- **DummyFormConsumer** which extends DummyForm

DummyForm will define the content of our form, our fields our widget etc

Models :

the model **th_dummy/models.py** :

```
class Dummy(Services):

    # put whatever you need  here
    # eg title = models.CharField(max_length=80)
    # but keep at least this one
    title = models.CharField(max_length=80)
    trigger = models.ForeignKey('TriggerService')

    class Meta:
        app_label = 'django_th'

    def __str__(self):
        return "%s" % (self.name)

    def show(self):
        return "My Dummy %s" % (self.name)
```

Key points :

- The model is related to TriggerService model
- The model uses the **app_label** to **django_th** meta, so the Trigger Happy will be added the table name

Service class :

at the beginning of the class **ServiceDummy** (from *th_dummy/my_dummy.py*) you will need to import the class of the third party application

the class *ServiceDummy* will extend *ServiceMgr* we've imported from *django_th.services.services*

This class is composed at least by 2 methods :

save_data :

we provider the following parms

- token - the token of the service
- trigger_id - the trigger id we handle,
- data - the data to store (title, url, content), provided by a "process_data" of another service

role : save the data to the *ServiceDummy*

return : a boolean True or False, if the save_data worked fine or not

If the service does not save data, it's the case of the module django-th-rss which just provides stuff and save nothing, you'll put *pass* to save_data as the body of your code

auth and callback :

If your service need an authentication, you'll need 2 new functions *auth* and *callback*

- *auth* will trigger the authentication to the third party application, the Oauth process in fact
- *callback* is triggered when the authentication is done and call by the third party application.

At this step the callback function store the oauth token to the dedicated dummy model

The complete code of this class :

```
# coding: utf-8
# add here the call of any native lib of python like datetime etc.
#
# add the python API here if needed
from external_api import CallOfApi

# django classes
from django.conf import settings
from logging import getLogger

# django_th classes
from django_th.services.services import ServicesMgr
```

```
from django_th.models import UserService, ServicesActivated

"""
    handle process with dummy
    put the following in settings.py

    TH_DUMMY = {
        'consumer_key': 'abcdefghijklmnopqrstuvwxyz',
    }

    TH_SERVICES = (
        ...
        'th_dummy.my_dummy.ServiceDummy',
        ...
    )

"""

logger = getLogger('django_th.trigger_happy')


class ServiceDummy(ServicesMgr):


    def __init__(self, ):
        self.dummy_instance = external_api.CallOfApi(
                settings.TH_DUMMY['consumer_key'], token)

    def read_data(self, token, trigger_id, date_triggered):
        """
            get the data from the service
            :param trigger_id: trigger ID to process
            :param date_triggered: the date of the last trigger
            :type trigger_id: int
            :type date_triggered: datetime
            :return: list of data found from the date_triggered filter
            :rtype: list
        """
        data = list()
        return cache.set('th_dummy_' + str(trigger_id), data)

    def save_data(self, token, trigger_id, **data):
        """
            let's save the data

            :param trigger_id: trigger ID from which to save data
            :param **data: the data to check to be used and save
            :type trigger_id: int
            :type **data:  dict
            :return: the status of the save statement
            :rtype: boolean
        """
        from th_dummy.models import Dummy
        status = False

        if token and data.get('link'):
            # get the data of this trigger
            trigger = Dummy.objects.get(trigger_id=trigger_id)
```

```
            # if the external service need we provide
            # our stored token and token secret then I do
            # token_key, token_secret = token.split('#TH#')

            title = ''
            title = (data.get('title') if data.get('title') else '')
                # add data to the external service
            item_id = self.dummy_instance.add(
                url=data['link'], title=title, tags=(trigger.tag.lower()))

            sentance = str('dummy {} created').format(data.get('link'))
            logger.debug(sentance)
            status = True
        else:
            logger.critical(
                "no token or link provided for trigger ID {} ".format(trigger_id))
            status = False
        return status

    def auth(self, request):
        """
            let's auth the user to the Service
        """
        request_token = super(ServiceDummy, self).auth(request)
        callback_url = self.callback_url(request, 'dummy')

        # URL to redirect user to, to authorize your app
        auth_url_str = '%s?oauth_token=%s&oauth_callback=%s'
        auth_url = auth_url_str % (self.AUTH_URL,
                                   request_token['oauth_token'],
                                   callback_url)

        return auth_url

    def callback(self, request):
        """
            Called from the Service when the user accept to activate it
        """
        kwargs = {'access_token': '', 'service': 'ServiceDummy',
                  'return': 'dummy'}
        return super(ServiceDummy, self).callback(request, **kwargs)
```

MIGRATIONS from 0.10.x to 0.11.x :

Nota : in the SQL queries below, I use CURRENT_TIMESTAMP because of Postgresql. Adapt it to your own RDBMS.

Django Trigger Happy tables :

To migrate enter,

```
python manage.py  migrate
```

if the migration complains that you've already created the table django_th_rss then check the follow :

```
select * from django_migrations ;
```

to find

11 | django_th | 0001_initial | 2015-06-10 10:00:00.977958+02

if you don't have it then do :

```
insert into django_migrations (app,name,applied) values('django_th','0001_initial',
↪CURRENT_TIMESTAMP);
```

then replay

```
python manage.py migrate
```

Django Trigger Happy Module tables :

Evernote :

if the migration complains that you've already created the table django_th_evernote then check it by :

```
select * from django_migrations ;
```

check that you dont have those record in the django_migrations table

```
select * from django_migrations ;

13 | th_evernote       | 0001_initial        | 2015-06-10 10:00:00.977958+02
```

if its not the case, then add the following by hand like that :

```
insert into django_migrations (app,name,applied) values('th_evernote','0001_initial',
↪CURRENT_TIMESTAMP);
```

Holidays :

if the migration complains that you've already created the table django_th_holidays then check it by :

```
select * from django_migrations ;
```

check that you dont have those record in the django_migrations table

```
select * from django_migrations ;

13 | th_holidays       | 0001_initial        | 2015-06-10 10:00:00.977958+02
```

if its not the case, then add the following by hand like that :

```
insert into django_migrations (app,name,applied) values('th_holidays','0001_initial',
↪CURRENT_TIMESTAMP);
```

Pocket :

if the migration complains that you've already created the table django_th_pocket then check it by :

```
select * from django_migrations ;
```

check that you dont have those record in the django_migrations table

```
select * from django_migrations ;

13 | th_pocket        | 0001_initial         | 2015-06-10 10:00:00.977958+02
```

if its not the case, then add the following by hand like that :

```
insert into django_migrations (app,name,applied) values('th_pocket','0001_initial',
↪CURRENT_TIMESTAMP);
```

Readability :

if the migration complains that you've already created the table django_th_readability then check it by :

```
select * from django_migrations ;
```

check that you dont have those record in the django_migrations table

```
select * from django_migrations ;

13 | th_readability   | 0001_initial         | 2015-06-10 10:00:00.977958+02
```

if its not the case, then add the following by hand like that :

```
insert into django_migrations (app,name,applied) values('th_readability','0001_initial
↪',CURRENT_TIMESTAMP);
```

Twitter :

if the migration complains that you've already created the table django_th_twitter then check it by :

```
select * from django_migrations ;
```

check that you dont have those record in the django_migrations table

```
select * from django_migrations ;

13 | th_twitter      | 0001_initial         | 2015-06-10 10:00:00.977958+02
```

if its not the case, then add the following by hand like that :

```
insert into django_migrations (app,name,applied) values('th_twitter','0001_initial',
↪CURRENT_TIMESTAMP);
insert into django_migrations (app,name,applied) values('th_twitter','0002_int_to_
↪bigint',CURRENT_TIMESTAMP);
```

before adding by hand the line below, check that the table django_th_twitter contains the column max_id and since_id as bigint and not just int

if that columns are not bigint add just this

```
insert into django_migrations (app,name,applied) values('th_twitter','0001_initial',
↪CURRENT_TIMESTAMP);
```

otherwise add this too

```
insert into django_migrations (app,name,applied) values('th_twitter','0002_int_to_
↪bigint',CURRENT_TIMESTAMP);
```

Table to drop :

with the last

```
python manage.py migrate
```

you will meet this message :

```
Running migrations:
  No migrations to apply.
  Your models have changes that are not yet reflected in a migration, and so won't be␣
↪applied.
  Run 'manage.py makemigrations' to make new migrations, and then re-run 'manage.py␣
↪migrate' to apply them.
The following content types are stale and need to be deleted:

    django_th | userprofile
```

answer yes as this one is not used at all

then play again

```
python manage.py migrate
```

thus the migration will skip that steps and will continue smoothly

For example a new RSS item is published, **Trigger Happy** will be able to automatically create a note on your Evernote account or create a bookmark to your own Wallabag or Pocket account and so on

CHAPTER 2

Description:

The goal of this project is to be independent from any other solution like IFTTT, CloudWork or others.

Thus you could host your own solution and manage your own triggers without depending any non-free solution.

With this project you can host triggers for you.

All you need is to have a hosting provider (or simply your own server ;)) who permits to use a manager of tasks like “cron” and, of course Python.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search