PonyTracker Documentation

Release 0.1

Élie Bouttier

Sep 27, 2017

Contents

PonyTracker is a small and easy-to-use github-like issue tracker.

It is made in Python using the Django framework.

Contents:

CHAPTER 1

Installation guide

Typography

Commands starting with `#` must be run as `root` user.

Commands starting with `$` must be run as `ponytracker` user.

Requirements

You need python. Versions 2.7, 3.2 or 3.4 (and probably others) are fine.

This installation guide install PonyTracker in a virtualenv. The corresponding packages names are `python-virtualenv` for python 2 and `python3-virtualenv` for python 3 (at least under Debian).

You need also `supervisord` to launch PonyTracker. You can install it with the package with the same name.

Installation

Clone and configuration

Be sure `/srv/www` exists:

```
# mkdir -p /srv/www
```

Create a dedicated user:

```
# useradd -r -m -d /srv/www/ponytracker ponytracker
```

The following commands are executed as `ponytracker` user:

```
# su ponytracker
$ cd /srv/www/ponytracker
```

Create the `log` and `www` directories:

```
$ mkdir log www
```

Clone the git repository and switch to the release branch:

```
$ git clone http://github.com/ponytracker/ponytracker
$ cd ponytracker # we are now in /srv/www/ponytracker/ponytracker
$ git checkout release
```

Copy the example configuration file:

```
$ cp ponytracker/local_settings.py.example ponytracker/local_settings.py
```

Set the `SECRET_KEY` value. You can generate a secret key with `openssl`:

```
$ openssl rand -base64 32
```

If you want email notifications, set `BASE_URL`, `EMAIL_HOST` (smtp relay) and `DEFAULT_FROM_EMAIL`.

If you want to use another database than the default one (SQLite), set `DATABASES`. You can find the syntax in the django documentation.

Set the static directory:

```
STATIC_ROOT = '/srv/www/ponytracker/www/static'
```

Create a virtualenv and activate it:

```
$ virtualenv env
$ source env/bin/activate
```

Install the requirements:

```
$ pip install -r requirements.txt
```

Initialize the database tables:

```
$ python manage.py migrate
```

Create an account for the administrator:

```
$ python manage.py createsuperuser
```

Install javascript libraries with bower:

```
$ python manage.py bower install
```

Collect static files in the `STATIC_DIR`:

```
$ python manage.py collectstatic --settings=ponytracker.local_settings
```

Run django

You can use `gunicorn` or `uwsgi`, both started by `supervisord`.

Using gunicorn

Install gunicorn:

```
$ pip install gunicorn
```

Create the new file `/etc/supervisor/conf.d/ponytracker.conf` containing:

```
[program:ponytracker]
command=/srv/www/ponytracker/ponytracker/env/bin/gunicorn ponytracker.wsgi:application
directory=/srv/www/ponytracker/ponytracker
environment=PATH="/srv/www/ponytracker/ponytracker/env/bin"
user=ponytracker
autostart=true
autorestart=true
redirect_stderr=true
```

Update the file `ponytracker/wsgi.py` to use local settings instead of default settings:

```
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "ponytracker.local_settings")
```

In order not to have conflicts during an update, you can tell *git* to ignore this modification:

```
$ git update-index --assume-unchanged ponytracker/wsgi.py
```

Using uwsgi

Install uwsgi:

```
$ pip install uwsgi
```

Create the new file `/etc/supervisor/conf.d/ponytracker.conf` containing:

```
[program:ponytracker]
command=/srv/www/ponytracker/ponytracker/env/bin/uwsgi --ini uwsgi.ini
directory=/srv/www/ponytracker/ponytracker
environment=PATH="/srv/www/ponytracker/ponytracker/env/bin"
environment=DJANGO_SETTINGS_MODULE="settings.local_settings"
user=ponytracker
autostart=true
autorestart=true
redirect_stderr=true
```

Create the new file `/srv/www/ponytracker/ponytracker/uwsgi.ini` containing:

```
[uwsgi]
chdir=/srv/www/ponytracker/ponytracker
module=ponytracker.wsgi:application
master=True
pidfile=/tmp/ponytracker.pid
vacuum=True
```

```
max-requests=5000
http-socket = 127.0.0.1:8000
```

Verify

Start `supervisord` to run the server:

```
$ service supervisor start
```

You can check that the server is listening on `localhost` on port `8000`:

```
$ lsof | grep LISTEN | grep ponytracker
```

You now need to install a real web server as a front-end. This documentation shows sample configuration files for both *apache* and *nginx*.

Front-end

Apache

```
<VirtualHost *:443>
    ServerName ponytracker.example.com
    ServerAdmin webmaster@example.com
    DocumentRoot /var/empty

    RewriteEngine on
    RewriteRule ^(.*)$ https://%{HTTP_HOST}$1 [R=301,L]
</VirtualHost>

<VirtualHost *:443>
    ServerName ponytracker.example.com
    ServerAdmin webmaster@example.com
    DocumentRoot /srv/www/ponytracker/www

    SSLEngine on
    SSLCertificateFile /etc/apache2/ssl/ponytracker.example.com.pem
    SSLCertificateKeyFile /etc/apache2/ssl/ponytracker.example.com-Key.pem
    SSLCACertificateFile /etc/apache2/ssl/ponytracker.example.com-CA.pem

    <Location /static>
        ProxyPass !
    </Location>

    <Location />
        ProxyPass http://127.0.0.1:8000/
        ProxyPassReverse http://127.0.0.1:8000/
    </Location>

    ErrorLog /srv/www/ponytracker/log/error.log
    CustomLog /srv/www/ponytracker/log/access.log combined
</VirtualHost>
```

nginx

Forthcoming...

Send email asynchronously with the celery worker

By default, email notifications are sent during processing of pages. This behaviour can slow down the user experience. It is recommended to use a separated thread to asynchronously send the emails.

PonyTracker is designed to use celery as a worker to send emails. In order to get celery working, you need a broker. This guide uses the `redis` broker but you can find how to install another broker in the celery documentation.

Install `redis`:

```
# aptitude install redis
```

Enter in the virtualenv and install python requirements:

```
# su ponytracker
$ cd /srv/www/ponytracker/ponytracker
$ source env/bin/activate
$ pip install celery[redis]
$ pip install django-celery
```

Add `djcelery` to your enabled applications in your local settings (`ponytracker/local_settings.py`):

```
INSTALLED_APPS += ('djcelery',)
```

Enable celery specific commands for the manage.py script by adding these lines in your local settings:

```
import djcelery
djcelery.setup_loader()
```

Tell celery to use your redis broker by adding the `BROKER_URL` in your local settings:

```
BROKER_URL = 'redis://localhost:6379/0'
```

Run the celery worker:

```
$ python manage.py celery worker --loglevel=info --settings=ponytracker.local_setting
```

Forthcomming: how to launch celery from supervisord.

Use LDAP authentication

The python package providing LDAP support need to be compiled. Don't worry, `pip` does it itself, but you need to install some requirements:

```
# apt-get install python-dev libldap2-dev libsasl2-dev libssl-dev
```

`libssl-dev` is required only if want to use a SSL connection to your LDAP server. `libsasl2-dev` is only required if you want to use SASL authentication.

Enter in the virtualenv and install needed packages:

```
# su ponytracker
$ cd /srv/www/ponytracker/ponytracker
$ source env/bin/activate
$ pip install python-ldap django-auth-ldap
```

Add `django_auth_ldap.backend.LDAPBackend` to your authentication back-ends in `ponytracker/local_settings.py`:

```
AUTHENTICATION_BACKENDS += (
    'django_auth_ldap.backend.LDAPBackend',
)
```

Configure the back-end by adding required variables in your local settings. You can find the documentation on the official website. A sample file is provided.

If you use `posixGroup`, import `PosixGroupType` instead of `GroupOfNamesType` and update the `AUTH_LDAP_GROUP_TYPE` variable.

Add the following line to synchronize your LDAP groups with django ones:

```
AUTH_LDAP_MIRROR_GROUPS = True
```

Setup emails answering

PonyTracker allow users to directly answer email in order to add a new comment to an issue. To enable this feature, add these two parameters in your configuration file:

```
REPLY_EMAIL = 'reply@ponytracker.example.com'
EMAIL_KEY = 'CHANGE ME'
```

Then, you need to handle all emails received on `REPLY_EMAIL` and do a post request on *http://ponytracker.example.com/api/email/recv/*‘ with them. The post request should contain the key `key` containing `EMAIL_KEY` value and the email as a Multipart-Encoded file named *email*.

For that, you can use a sieve filter with the filter plugin.

To activate the filter with dovecot, copy the file `tools/ponytracker-sieve-filter` to `/usr/lib/dovecot/sieve-filter/ponytracker`. Be sure to activate the `vnd.devocot.filter` in dovecot configuration, usually in `/etc/dovecot/conf.d/90-sieve.conf`:

```
sieve_extensions = +vnd.dovecot.execute
```

Then, you can use a similar sieve script:

```
require ["fileinto", "imap4flags", "vnd.dovecot.filter"];

if address :matches "To" "reply*@ponytracker.example.com" {
  filter "ponytracker" "EMAIL_KEY@http://ponytracker.example.com/api/email/recv/";
  addflag "\\Seen";
  fileinto "Trash";
}
```

CHAPTER 2

Upgrade guide

The upgrade commands are executed as `ponytracker` user:

```
# su ponytracker
$ cd /srv/www/ponytracker
```

Activate the virtualenv:

```
$ source env/bin/activate
```

Enter in the repository directory:

```
$ cd ponytracker # we are now in /srv/www/ponytracker/ponytracker
```

Upgrade the files using `git`:

```
$ git pull -u master release
```

Install all new dependencies and upgrade previous ones:

```
$ pip install -r requirements.txt --upgrade
```

Be sure to use the correct configuration file each time you run the `manage.py` script by setting the `DJANGO_SETTING_MODULE` environment variable:

```
$ export DJANGO_SETTINGS_MODULE=ponytracker.local_settings
```

Collect static files to the `STATIC_DIR`:

```
$ python manage.py collectstatic
```

Apply database migrations:

```
$ python manage.py migrate
```

You can now restart ponytracker by restarting `gunicorn` or `uwsgi` depending of your installation. Do not forget to restart the celery worker too if you have installed it.

CHAPTER 3

User guide

CHAPTER 4

Developer guide

CHAPTER 5

Indices and tables

- genindex
- modindex
- search