nodeshot Documentation

Release alpha

Federico Capoano

Jun 01, 2017

Contents

Nodeshot is an open source platform for **crowdsourcing georeferenced data**.

Its goal it's to provide some robust tools to build a modern customized crowdsourcing web application quickly.

It is designed to be modular, flexible, configurable and extensible.

This documentation is a work in progress.

CHAPTER 1

Code

The code is hosted at **Github**: https://github.com/ninuxorg/nodeshot.

CHAPTER 2

Mailing list

If you have any issue or you want to follow the development of this project you can reach us at our Mailing List.

For past activities consult The Nodeshot Archives.

CHAPTER 3

Online instances

You can see a few running instances at the following URLs:

- https://ninux.nodeshot.org/ (english)
- https://opendata.publicwifi.it/ (english)
- https://piemontespot.nodeshot.org/ (italian)

CHAPTER 4

Contents

Development Installation

> **Warning:** This installation guide is designed for debian based operating systems.
>
> Other Linux distributions are good as well but the name of some packages may vary.

This document describes how to install nodeshot for **development**.

We are assuming you are executing commands as a **sudoer** user but **not root**.

Related pages: *Contribute to nodeshot*.

Install dependencies

First of all, update your apt cache:

```
sudo apt-get update --fix-missing
```

Install dependencies and development packages:

```
# dependencies
sudo apt-get install python-software-properties software-properties-common build-
↪essential libxml2-dev python-setuptools python-virtualenv python-dev binutils␣
↪libjson0-dev libjpeg-dev libffi-dev libpq-dev
# dev packages
sudo apt-get install wget git
```

Install postgresql, postgis and geospatial libraries:

```
sudo apt-get install postgis* libproj-dev gdal-bin libgdal1-dev python-gdal
```

Create database

Become `postgres` user:

```
sudo su postgres
```

Create database, create required postgresql extensions, create a superuser:

```
createdb nodeshot
psql nodeshot
CREATE EXTENSION postgis;
CREATE EXTENSION hstore;
CREATE USER nodeshot WITH PASSWORD 'your_password';
ALTER USER nodeshot SUPERUSER;
```

exit (press CTRL+D) and go back to being your user:

```
exit
```

Install python packages

First of all, install virtualenvwrapper (systemwide):

```
sudo pip install virtualenvwrapper
```

virtualenvwrapper needs some initialization before you can use its shortcuts:

```
echo 'source /usr/local/bin/virtualenvwrapper.sh' >> ~/.bashrc
source ~/.bashrc
```

Create a **python virtual environment**, which is a self-contained python installation which will contain all the python packages required by nodeshot:

```
mkvirtualenv nodeshot
```

Update the basic python utilities:

```
pip install -U setuptools pip wheel
```

Clone your fork in your favourite location (`/home/<user>` or `/var/www`), have you forked nodeshot, right?

```
git clone git@github.com:<YOUR-FORK>/nodeshot.git
cd nodeshot
```

Replace `<YOUR-FORK>` with your github username (be sure to have forked nodeshot first).

Install the required python packages:

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

Finally install nodeshot with:

```
python setup.py develop
```

Create the development project, be sure it's called **dev**:

```
nodeshot startproject dev && cd dev
```

Project configuration

Open `settings.py`:

```
vim dev/settings.py
```

And edit the following settings:

- `DOMAIN`: set localhost
- `DATABASE['default']['USER']`: set nodeshot
- `DATABASE['default']['PASSWORD']`: set the password chosen during the *Create database* step

Create the database tables and initial data:

```
# will prompt you to create a superuser, proceed!
./manage.py migrate --no-initial-data && ./manage.py loaddata initial_data
```

Run the development server:

```
./manage.py runserver
```

Alternatively, if you need to reach the dev server for other hosts on the same LAN, you can setup the development server to listen on all the network interfaces:

```
./manage.py runserver 0.0.0.0:8000
```

Now you can open your browser at http://localhost:8000/ or at http://localhost:8000/admin/.

How to setup the test environment

The `/test` directory contains a nodeshot project called `ci` (stands for continuous integration) that is needed to run automated tests (unit tests, functional tests and regression tests).

Install the hstore extension on template1 according to how to run tests with django-hstore:

```
sudo su postgres
psql template1 -c 'CREATE EXTENSION hstore;'
exit
```

Do a `cd` into the `/test` dir:

```
cd /[PATH-TO-NODESHOT-REPO]/tests
```

Create a `local_settings.py` file:

```
cp ci/local_settings.example.py ci/local_settings.py
```

Ensure your virtualenv is activated:

```
workon nodeshot
```

Run all the tests with:

```
./runtests.py --keepdb
```

The `keepdb` option allows to avoid recreating the test database at each run, **hence saving precious time**.

If you want to speed up tests even more, tweak your local postgresql configuration by setting these values:

```
# /etc/postgresql/9.1/main/postgresql.conf
# only for development!
fsync = off
synchronous_commit = off
full_page_writes = off
```

Test specific modules

Each module has its own tests, so you can test one module at time:

```
python manage.py test --keepdb nodeshot.core.nodes
```

You can also test more modules:

```
python manage.py test --keepdb nodeshot.core.nodes nodeshot.core.layers nodeshot.core.
↪cms
```

Test coverage

Install coverage package:

```
pip install coverage
```

Run test coverage and get a textual report:

```
coverage run --source=nodeshot runtests.py --keepdb && coverage report
```

Calculate test coverage for specific modules:

```
coverage run --source=nodeshot.core.nodes ./manage.py test --keepdb nodeshot.core.
↪nodes && coverage report
```

Measure time spent running tests

Automated tests that involve database calls and/or HTTP requests can quickly become slow.

Slow tests mean low productivity, especially if you are used to **Test Driven Development**.

For this reason, if you notice that some tests are slow, you have two additional options to measure test executition times, **find out which tests are slow** and refactor them.

Time option

`--time` will measure how much time is needed to execute each test class.

Detailed option

`--time --detailed` will measure how much time is needed to execute each test class **and** each single test.

Examples

Here's a couple of examples:

```
# general measurement
./runtests.py --keepdb --time

# detailed measurement
./runtests.py --keepdb --time --detailed

# detailed measurement for a specific test class
python manage.py test --keepdb --time --detailed nodeshot.core.nodes
```

How to build the documentation

Building the documentation locally is useful for several reasons:

- you can read it offline
- you can edit it locally
- you can preview the changes locally before sending any pull request

So let's **build the docs**!

Install sphinx:

```
workon nodeshot
pip install sphinx
```

Do a `cd` into the `/docs` dir:

```
cd /[PATH-TO-NODESHOT-REPO]/docs
```

Now build the docs with:

```
make html
```

Quite some html files have been created, you can browse those HTML files in a web browser and it should work.

The format used in the docs is **reStructured Text** while the python package used is **python-sphinx**.

Read more information about Sphinx and reStructured Text.

Contribute

If you intend to contribute to nodeshot, be sure to read *How to contribute to nodeshot*.

Automated Production Installation

This section describes how to perform a quick install of Nodeshot on **Ubuntu / Debian systems**.

> **Warning:** This procedure has been tested on clean installs of **Debian 7**, **Ubuntu 13.10** and **Ubuntu 14.04 LTS**.
>
> If you try it on a server where other applications are running you might incur in some errors.
>
> The most typical would be having the port 80 already in use by Apache.
>
> In that case, you should consider using the *Manual Production Installation* procedure in order to install according to your needs.

Prerequisites

First of all, we need to install the Fabric Python library.

To install fabric, you need to have pip installed on your system. See how to install pip first.

Proceed to install Fabric:

```
pip install fabric
```

More detailed instructions about Fabric installation can be found here.

Download nodeshot-fabfile

Download the archive:

- *tarball*: https://github.com/ninuxorg/nodeshot-fabfile/tarball/master
- *zip*: https://github.com/ninuxorg/nodeshot-fabfile/archive/master.zip

eg:

```
wget https://github.com/ninuxorg/nodeshot-fabfile/archive/master.zip -O nodeshot-
↪fabfile.zip
unzip nodeshot-fabfile.zip
rm nodeshot-fabfile.zip
cd nodeshot-fabfile-master
```

Alternatively you can use git:

```
sudo apt-get install git
git clone https://github.com/ninuxorg/nodeshot-fabfile.git
cd nodeshot-fabfile
```

Start installing

> **Warning:** We suggest to install on a clean virtual machine

Start the fabfile script with:

```
fab install -H <remote_host> -u <user> -p <password>
```

<user> should be either a sudoer or root.

The install procedure will start, asking you to insert the parameters that will customize your nodeshot instance:

```
stefano@stefano:~/django/nodeshot/INSTALL$ fab install -H nodeshot-deploy-example.cineca.it -u user -p password
[nodeshot-deploy-example.cineca.it] Executing task 'install'
Set install directory ( including trailing slash ): [/var/www/]
Set project name: [myproject] citySDK
Server name: nodeshot-deploy-example.cineca.it
Set database user: [nodeshot]
Set database user password: nodeshot
```

These are the informations you will have to supply to the install procedure:

Install directory: the directory where Nodeshot will be installed (default: /var/www/)

Project name: the name for your project (default: myproject), **avoid using nodeshot, test, or other existing python packages or the installer will break**

Server name: the FQDN of your server (no default)

Database user: postgres owner of Nodeshot DB (default: nodeshot)

Database user password: password for postgres owner of Nodeshot DB (default generated randomly)

Next, you will have to supply the details for the SSL certificate that will be used for serving Nodeshot over HTTPS:

```
************************************
Please insert SSL certificate details...
************************************
[nodeshot-deploy-example.cineca.it] run: mkdir -p /tmp/nodeshot_install
[nodeshot-deploy-example.cineca.it] run: openssl req -new -x509 -nodes -days 365 -out server.crt -keyout server.key
[nodeshot-deploy-example.cineca.it] out: Generating a 1024 bit RSA private key
[nodeshot-deploy-example.cineca.it] out: ...................++++++
[nodeshot-deploy-example.cineca.it] out: .................++++++
[nodeshot-deploy-example.cineca.it] out: writing new private key to 'server.key'
[nodeshot-deploy-example.cineca.it] out: -----
[nodeshot-deploy-example.cineca.it] out: You are about to be asked to enter information that will be incorporated
[nodeshot-deploy-example.cineca.it] out: into your certificate request.
[nodeshot-deploy-example.cineca.it] out: What you are about to enter is what is called a Distinguished Name or a DN.
[nodeshot-deploy-example.cineca.it] out: There are quite a few fields but you can leave some blank
[nodeshot-deploy-example.cineca.it] out: For some fields there will be a default value,
[nodeshot-deploy-example.cineca.it] out: If you enter '.', the field will be left blank.
[nodeshot-deploy-example.cineca.it] out: -----
[nodeshot-deploy-example.cineca.it] out: Country Name (2 letter code) [AU]:IT
[nodeshot-deploy-example.cineca.it] out: State or Province Name (full name) [Some-State]:Italy
[nodeshot-deploy-example.cineca.it] out: Locality Name (eg, city) []:Rome
[nodeshot-deploy-example.cineca.it] out: Organization Name (eg, company) [Internet Widgits Pty Ltd]:Cineca
[nodeshot-deploy-example.cineca.it] out: Organizational Unit Name (eg, section) []:CitySDK dev
[nodeshot-deploy-example.cineca.it] out: Common Name (e.g. server FQDN or YOUR name) []:nodeshot-deploy-example.cineca.it
[nodeshot-deploy-example.cineca.it] out: Email Address []:admin@nodeshot-deploy-example.cineca.it
```

That's all you have to do: the installation process will start.

It will take care of installing package dependencies, creating a python virtualenv, configuring the webserver and the all the other bits needed to run Nodeshot.

The installation will take about 5-10 minutes to complete. As final step, it will start all services and leave you with a full running version of Nodeshot.

A message will remind you to change the default admin account password:

```
Starting Nodeshot server...
[nodeshot-deploy-example.cineca.it] run: service nginx restart && supervisorctl restart all
[nodeshot-deploy-example.cineca.it] out: Restarting nginx: nginx.
[nodeshot-deploy-example.cineca.it] out: celery-beat: stopped
[nodeshot-deploy-example.cineca.it] out: uwsgi: stopped
[nodeshot-deploy-example.cineca.it] out: celery-beat: started
[nodeshot-deploy-example.cineca.it] out: uwsgi: started
[nodeshot-deploy-example.cineca.it] out: celery: started

Nodeshot server started
Cleaning installation directory...
[nodeshot-deploy-example.cineca.it] run: rm -rf /tmp/nodeshot_install
Installation completed
Cleaning installation directory...
[nodeshot-deploy-example.cineca.it] run: rm -rf /tmp/nodeshot_install

INSTALLATION COMPLETED !

############################################################
                    WARNING:
 Superuser is currently set as 'admin' with password 'admin'
 Logon on nodeshot-deploy-example.cineca.it/admin and change it
############################################################

Done.
Disconnecting from nodeshot-deploy-example.cineca.it... done.
```

Updating an existing instance

To run an update do:

```
fab update -H <remote_host> -u <user> -p <password>
```

If you need to specify parameters without the need to be prompted do:

```
fab update:use_defaults=True,project_name=<project_name> -H <remote_host> -u <user> -
↪p <password>
```

You could also set a different `root_dir` with:

```
fab update:use_defaults=True,root_dir=/custom/path/,project_name=<project_name> -H
↪<remote_host> -u root -p <password>
```

Manual Production Installation

> **Warning:** This document describes how to install nodeshot **Ubuntu Server 14.04 LTS** and **Debian 7** (other versions of ubuntu and debian should work as well with minor tweaks).
>
> Other Linux distributions are good as well but the name of some packages may vary.

If you already have the required dependencies installed you can skip to *Install python packages* and follow until *Project configuration*.

If you are installing for a **production environment** you need to follow all the instructions including *Production instructions*.

Required dependencies:

- Postgresql 9.1+
- Geospatial libraries and plugins (GEOS, Proj, Postgresql Contrib, ecc)
- Postgis 2.0+
- Python 2.7+
- Python Libraries (Virtualenv, setuptools, python-dev)

Required python packages:

- Django 1.8
- Django Rest Framework 2.4

A full list is available in the requirements.txt file.

Recommended stack for production environment:

- **Nginx**: main web server
- **uWSGI**: application server (serves requests to django)
- **Supervisor**: daemon process manager (used to manage uwsgi, celery and celery-beat)
- **Redis**: in memory key-value store (used as a message broker and cache storage)
- **Postfix**: SMTP server (send mails to users)

Install dependencies

First of all I suggest to become `root` to avoid typing sudo each time:

```
sudo -s
```

First of all, update your apt cache:

```
sudo apt-get update --fix-missing
```

Install dependencies:

```
sudo apt-get install python-software-properties software-properties-common build-
↪essential libxml2-dev python-setuptools python-virtualenv python-dev binutils␣
↪libjson0-dev libjpeg-dev libffi-dev libpq-dev
```

Install postgresql, postgis and geospatial libraries:

```
sudo apt-get install postgis* libproj-dev gdal-bin libgdal1-dev python-gdal
```

Create database

Set `postgres` user password:

```
passwd postgres
```

Become `postgres` user:

```
su postgres
```

Create database, create required postgresql extensions, create a user and grant all privileges to the newly created DB:

```
createdb  nodeshot
psql nodeshot
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_topology;
CREATE EXTENSION hstore;
CREATE USER nodeshot WITH PASSWORD 'your_password';
GRANT ALL PRIVILEGES ON DATABASE "nodeshot" to nodeshot;
```

exit (press CTRL+D) and go back to being root:

```
exit
```

Install python packages

First of all, install virtualenvwrapper:

```
pip install virtualenvwrapper
mkdir /usr/local/lib/virtualenvs
echo 'export WORKON_HOME=/usr/local/lib/virtualenvs' >> /usr/local/bin/
↪virtualenvwrapper.sh
echo 'source /usr/local/bin/virtualenvwrapper.sh' >> ~/.bashrc
echo 'source /usr/local/bin/virtualenvwrapper.sh' >> ~/.bash_profile
source ~/.bashrc
```

Now you can create a **python virtual environment**, which is a self-contained python installation which will contain all the python packages required by nodeshot:

```
mkvirtualenv nodeshot
```

Update the basic python utilities:

```
pip install -U setuptools pip wheel
```

Now, if you are installing for **production** you should install nodeshot and its dependencies with:

```
pip install https://github.com/ninuxorg/nodeshot/tarball/master
```

Now create the directory structure that will contain the project, a typical web app is usually installed in `/var/www/`:

```
mkdir -p /var/www/nodeshot && cd /var/www/
```

Create the nodeshot settings folder:

```
nodeshot startproject <myproject> nodeshot
cd nodeshot
chown -R <user>:www-data .  # set group to www-data
adduser www-data <user>
chmod 775 . log <myproject> <myproject>/media # permit www-data to write logs, pid␣
↪files and static directory
chmod 750 manage.py <myproject>/*.py  # do not permit www-data to write on python␣
↪files*
```

Replace <myproject> with your project name. **Avoid names which are used by existing python packages (eg: test) and avoid calling it nodeshot (that's already taken by nodeshot itself)**, prefer a short and simple name, for example: **ninux**, **yourcommunity**, **yourdomain**.

Replace <user> with your current non-root user (the one which created the virtualenv).

Project configuration

Open settings.py:

```
vim <myproject>/settings.py
```

And edit the following settings with your configuration:

- DOMAIN (domain or ip address)
- DATABASE (host, db, user and password)

Now setup the database:

```
exit  # go back being non-root
# will prompt you to create a superuser, proceed!
./manage.py migrate --no-initial-data && ./manage.py loaddata initial_data
```

Copy static assets (javascript, css, images):

```
./manage.py collectstatic
```

Production instructions

In production you will need reliable instruments, we recommend the following software stack:

- **Nginx**: main web server
- **uWSGI**: application server (serves requests to django)
- **Supervisor**: daemon process manager (used to manage uwsgi, celery and celery-beat)
- **Redis**: in memory key-value store (used as a message broker and cache storage)
- **Postfix**: SMTP server (send mails to users)

Nginx

Nginx is the recommended webserver for nodeshot.

Alternatively you could also use any other webserver like apache2 or lighthttpd but it won't be covered in this doc.

You can install from the system packages with the following command:

```
sudo -s  # become root again
apt-get install nginx-full nginx-common openssl zlib-bin
```

Create a temporary self signed SSL certificate (or install your own one if you already have it):

```
mkdir /etc/nginx/ssl
cd /etc/nginx/ssl
openssl req -new -x509 -nodes -out server.crt -keyout server.key
```

Copy `uwsgi_params` file:

```
cp /etc/nginx/uwsgi_params /etc/nginx/sites-available/
```

Create public folder:

```
mkdir /var/www/nodeshot/public_html
```

Create site configuration (replace `nodeshot.yourdomain.com` with your domain):

```
vim /etc/nginx/sites-available/nodeshot.yourdomain.com
```

Paste this configuration and tweak it according to your needs:

```
server {
    listen 443 ssl;  # ipv4
    #listen  [::]:443 ssl; # ipv6

    root /var/www/nodeshot/public_html;
    index index.html index.htm;

    # error log
    error_log /var/www/nodeshot/log/nginx.error.log error;

    # Make site accessible from hostanme
    # change this according to your domain/hostanme
    server_name nodeshot.yourdomain.com;

    # set client body size #
    client_max_body_size 5M;

    ssl on;
    ssl_certificate ssl/server.crt;
    ssl_certificate_key ssl/server.key;
    # optimizations
    ssl_session_cache shared:SSL:20m;
    ssl_session_timeout 10m;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDH+AESGCM:ECDH+AES256:ECDH+AES128:DH+3DES:!ADH:!AECDH:!MD5;
    add_header Strict-Transport-Security "max-age=31536000";
    add_header X-Content-Type-Options nosniff;

    location @uwsgi {
        uwsgi_pass 127.0.0.1:3031;
        include uwsgi_params;
        uwsgi_param HTTP_X_FORWARDED_PROTO https;
    }

    location / {
        try_files /var/www/nodeshot/maintenance.html $uri @uwsgi;
    }

    location /static {
        alias /var/www/nodeshot/<myproject>/static/;
    }

    location /media {
        alias /var/www/nodeshot/<myproject>/media/;
```

```
    }
}

server {
    listen 80;  # ipv4
    #listen [::]:80;  # ipv6

    # Make site accessible from hostanme on port 80
    # change this according to your domain/hostanme
    server_name nodeshot.yourdomain.com;

    # redirect all requests to https
    return 301 https://$host$request_uri;
}
```

Keep replacing `<myproject>` with the project name chosen at the beginning.

Create a symbolic link to sites-enabled directory:

```
ln -s /etc/nginx/sites-available/nodeshot.yourdomain.com /etc/nginx/sites-enabled/
```

Test config, ensure it does not fail:

```
service nginx configtest
```

Now you can reload nginx server:

```
service nginx restart
```

uWSGI

uWSGI is a performant and scalable application server written in C.

We will use it to serve requests to the nodeshot django apps.

Install the latest version via pip:

```
# deactivate python virtual environment
deactivate
# install uwsgi globally
pip install uwsgi
```

Create a new ini configuration file:

```
vim /var/www/nodeshot/uwsgi.ini
```

Paste this config (keep replacing `<myproject>` with the project name chosen at the beginning):

```
[uwsgi]
chdir=/var/www/nodeshot
module=<myproject>.wsgi:application
master=True
pidfile=/var/www/nodeshot/uwsgi.pid
socket=127.0.0.1:3031
processes=2
harakiri=20
max-requests=5000
```

```
vacuum=True
home=/usr/local/lib/virtualenvs/nodeshot
enable-threads=True
env=HTTPS=on
buffer-size=8192
```

Redis

Redis is used as message broker for *Celery* and as *Cache Storage*.

If you are using **debian 7**, you need to install a more recent version from a third party package maintainer:

```
echo "deb http://packages.dotdeb.org wheezy all" > /etc/apt/sources.list.d/dotdeb.list
echo "deb-src http://packages.dotdeb.org wheezy all" >> /etc/apt/sources.list.d/
↪dotdeb.list
wget http://www.dotdeb.org/dotdeb.gpg
apt-key add dotdeb.gpg
apt-get update
apt-get install redis-server
```

otherwise you can just run:

```
apt-get install redis-server
```

Install celery bindings in your virtual environment:

```
workon nodeshot  # activates virtualenv again
pip install -U celery[redis]
```

Change the `DEBUG` setting to `False`, leaving it to `True` **might lead to poor performance or security issues**:

```
vim /var/www/nodeshot/<myproject>/settings.py
# set DEBUG to False
DEBUG = False
# save and exit
```

You might encounter an issue in the Redis log that says: "Can't save in background: fork: Cannot allocate memory", in that case run this command:

```
echo 1 > /proc/sys/vm/overcommit_memory
```

Restart redis and ensure is running:

```
service redis-server restart
service redis-server status
```

Supervisor

We will use Supervisor as a process manager. Install it via your package system (or alternatively via pip):

```
apt-get install supervisor
```

Create new config file:

```
vim /etc/supervisor/conf.d/uwsgi.conf
```

Save this in `/etc/supervisor/conf.d/uwsgi.conf`:

```
[program:uwsgi]
user=www-data
directory=/var/www/nodeshot
command=uwsgi --ini uwsgi.ini
autostart=true
autorestart=true
stopsignal=INT
redirect_stderr=true
stdout_logfile=/var/www/nodeshot/log/uwsgi.log
stdout_logfile_maxbytes=30MB
stdout_logfile_backups=5
```

Repeat in a similar way for celery:

```
vim /etc/supervisor/conf.d/celery.conf
```

And paste (replace `<myproject>` with the project name chosen at the beginning):

```
[program:celery]
user=www-data
directory=/var/www/nodeshot
command=/usr/local/lib/virtualenvs/nodeshot/bin/celery -A <myproject> worker -l info
autostart=true
autorestart=true
redirect_stderr=true
stdout_logfile=/var/www/nodeshot/log/celery.log
stdout_logfile_maxbytes=30MB
stdout_logfile_backups=10
startsecs=10
stopwaitsecs=600
numprocs=1
```

Now repeat in a similar way for celery-beat:

```
vim /etc/supervisor/conf.d/celery-beat.conf
```

And paste (replace `<myproject>` with the project name chosen at the beginning):

```
[program:celery-beat]
user=www-data
directory=/var/www/nodeshot
command=/usr/local/lib/virtualenvs/nodeshot/bin/celery -A <myproject> beat -s ./
↪celerybeat-schedule -l info
autostart=true
autorestart=true
redirect_stderr=true
stdout_logfile=/var/www/nodeshot/log/celery-beat.log
stdout_logfile_maxbytes=30MB
stdout_logfile_backups=10
startsects=10
numprocs=1
```

Then run:

```
rm /var/www/nodeshot/log/*.log  # reset logs
supervisorctl update
```

You can check the status with:

```
supervisorctl status
```

And you can also use other commands like start, stop and restart.

Postfix

Postfix is needed to send emails. By default postfix is configured to accept local connections only. It is better to leave this default config unchanged to avoid spam, unless you know what you are doing.

To have a working SMTP server in the least possible steps follow this procedure:

1. install postfix:

```
apt-get install postfix
```

2. open configuration in editor:

```
vim /etc/postfix/main.cf
```

3. disable TLS:

```
smtpd_use_tls=no
```

4. set `myhostname`:

```
myhostname = nodeshot.yourdomain.com
```

5. add your hostname to `destination`:

```
mydestination = localhost.localdomain, localhost, nodeshot.yourdomain.com
```

6. save changes and restart postfix:

```
service postfix restart
```

Restart all processes

Restart all the processes to reload the new configurations:

```
service nginx restart && supervisorctl restart all
```

You should be done!

Test your installation and if everything works as expected.

Support

If you have any issue and you need support reach us at our Mailing List.

Contribute

Warning: Don't wanna be a sucker right? Follow all the steps in this guide and everybody will be happy :-)

This document describes **how to contribute to nodeshot**.

1. Checkout open issues on github

The list of issues on Github is a great place to start looking for things to do.

2. Join the Mailing List

It would be great if you announced your intentions in the Mailing List.

That way we can **coordinate** and hopefully **support** you if you have questions.

3. Fork the github repo

Fork the nodeshot repository!

That's where you can work on your changes before pushing them upstream.

4. Install nodeshot for development

Follow the procedure described in *Development Installation*.

5. Learn how to run unit tests

Learn how to run unit tests in *How to setup the test environment*.

6. Follow PEP8, Style Guide for Python Code

Before writing any line of code, please ensure you have read and understood PEP 8 Style Guide for Python Code.

When more people are writing code **it is very important to stay consistent**.

7. Start writing code

Now you can finally start writing code!

8. Write tests for your code

Whether you are fixing a bug or adding a feature to an existing module, you should ensure that whenever a behaviour is changed there is an automated test that verifies that the code you wrote is behaving as expected.

More information about writing tests for django apps.

9. Ensure tests pass and coverage is not under 90%

Ensure that all the tests pass and that test coverage is not under 90%.

More info on *Test coverage*.

10. Document your changes

If you are adding features to modules or changing the default behaviour ensure to document your changes by editing the files in the `/docs` folder.

Read more about this topic in *How to build the documentation*.

11. Open pull request

Now you can finally open a **pull request** on **github** for review.

Optionally, you could open a pull request right after the first commit, so that the participants can review your commits as you push them.

12. Acknowledge Continuous Integration Testing

Each time commits are sent to the master branch or are added to a pull request, the test suite is automatically run on **travis-ci.org**, the result is shown in the **"build status"** which can either be *failed* or *passed*.

You can check the build status at travis-ci.org.

13. Adding features in separate modules

If you plan to add dramatic new features to nodeshot, it might better to explore the possibility of writing a new python package in a separate repository.

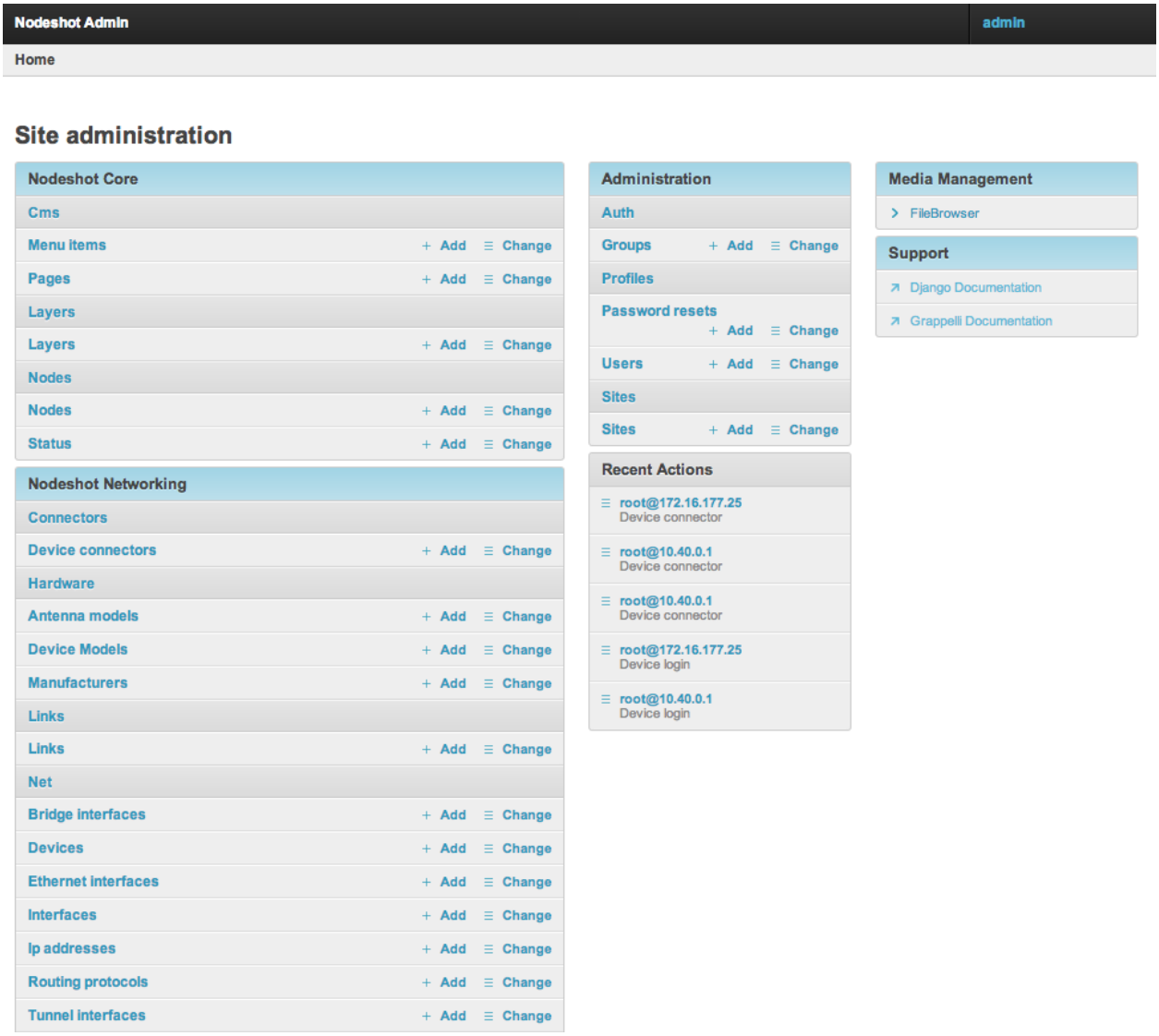
Find more information on How to write reusable apps.

Administration interface

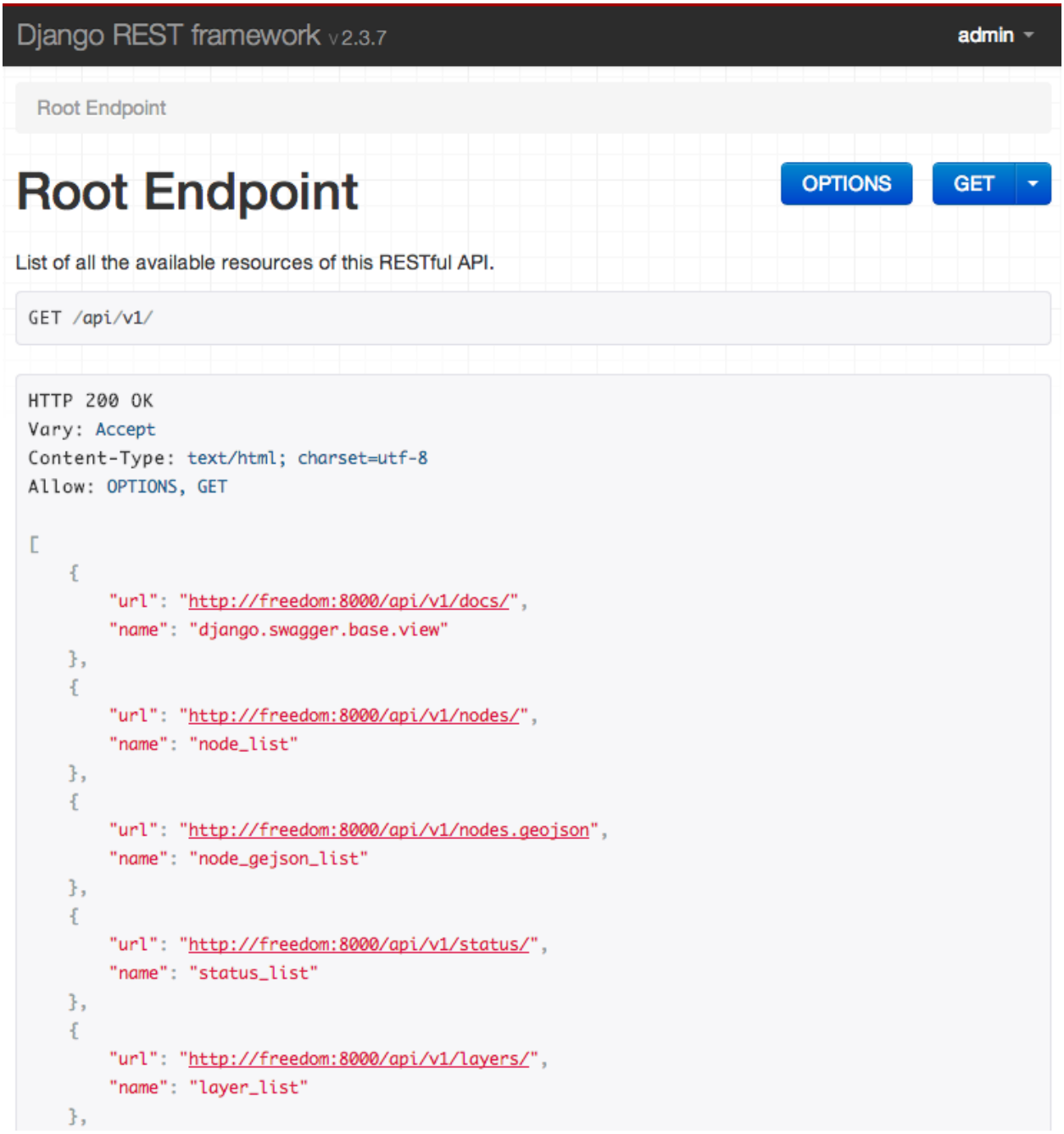
Nodeshot provides an Admin interface with advanced features which is built for administrators.

To open the admin interface open your browser at **http://localhost:8000/admin**.

Replace *http://localhost:8000* with your actual hostname.


Nodeshot Admin
admin
Home
Site administration
Nodeshot Core
Cms
Menu items + Add Change
Pages + Add Change
Layers
Layers + Add Change
Nodes
Nodes + Add Change
Status + Add Change
Nodeshot Networking
Connectors
Device connectors + Add Change
Hardware
Antenna models + Add Change
Device Models + Add Change
Manufacturers + Add Change
Links
Links + Add Change
Net
Bridge interfaces + Add Change
Devices + Add Change
Ethernet interfaces + Add Change
Interfaces + Add Change
Ip addresses + Add Change
Routing protocols + Add Change
Tunnel interfaces + Add Change
Administration
Auth
Groups + Add Change
Profiles
Password resets + Add Change
Users + Add Change
Sites
Sites + Add Change
Recent Actions
root@172.16.177.25
Device connector
root@10.40.0.1
Device connector
root@10.40.0.1
Device connector
root@172.16.177.25
Device login
root@10.40.0.1
Device login
Media Management
FileBrowser
Support
Django Documentation
Grappelli Documentation

Self documented RESTful API



Nodeshot provides a JSON RESTful API to manage most of the data in its database.

The API is **self-documented**, **browsable** and has two levels of documentation.

By default the API is reachable at **http://localhost:8000/api/v1/**.

Replace *http://localhost:8000* with your actual hostname.

Settings

`nodeshot.core.api` is enabled by default in `nodeshot.conf.settings.INSTALLED_APPS`.

These are the available customizable settings:

- `NODESHOT_API_PREFIX`
- `NODESHOT_API_APPS_ENABLED`

NODESHOT_API_PREFIX

default: `api/v1/`

The API URL prefix.

The following example will expose the API root to **http://localhost:8000/**:

```
NODESHOT_API_PREFIX = ""
```

NODESHOT_API_APPS_ENABLED

default:

```
[
    'nodeshot.core.nodes',
    'nodeshot.core.layers',
    'nodeshot.core.cms',
    'nodeshot.community.profiles',
    'nodeshot.community.participation',
    'nodeshot.community.notifications',
    'nodeshot.community.mailing',
    'nodeshot.networking.net',
    'nodeshot.networking.links',
    'nodeshot.networking.services',
    'nodeshot.interop.open311',
    'nodeshot.ui.default.api'
]
```

Each nodeshot django app contains some API resources, this setting tells the API which of those resources should be enabled. By default all of them are enabled.

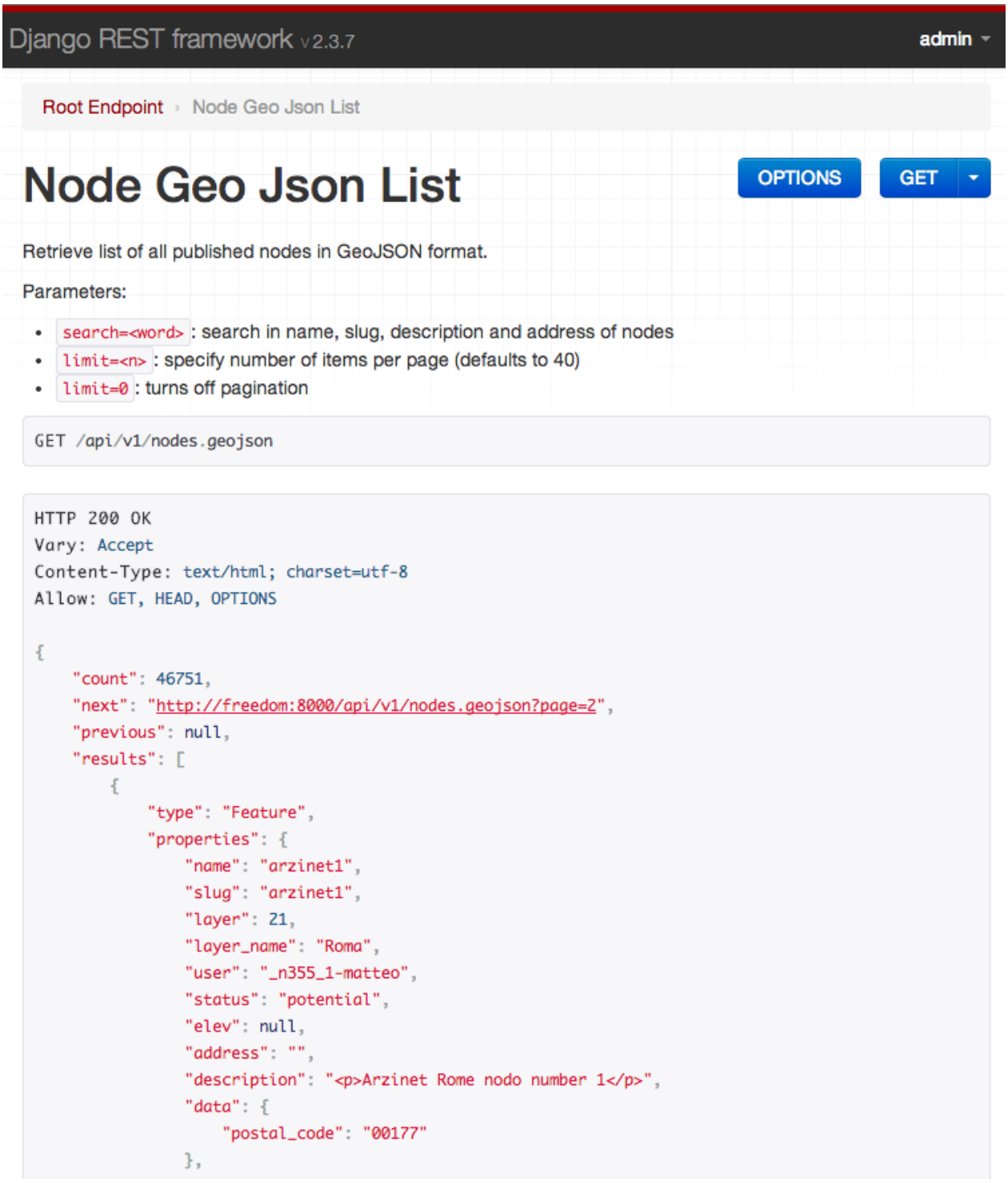
The following example enables only the API resources of the main modules:

```
# settings.py

NODESHOT_API_APPS_ENABLED = [
    'nodeshot.core.nodes',
    'nodeshot.core.layers',
    'nodeshot.core.cms'
]
```

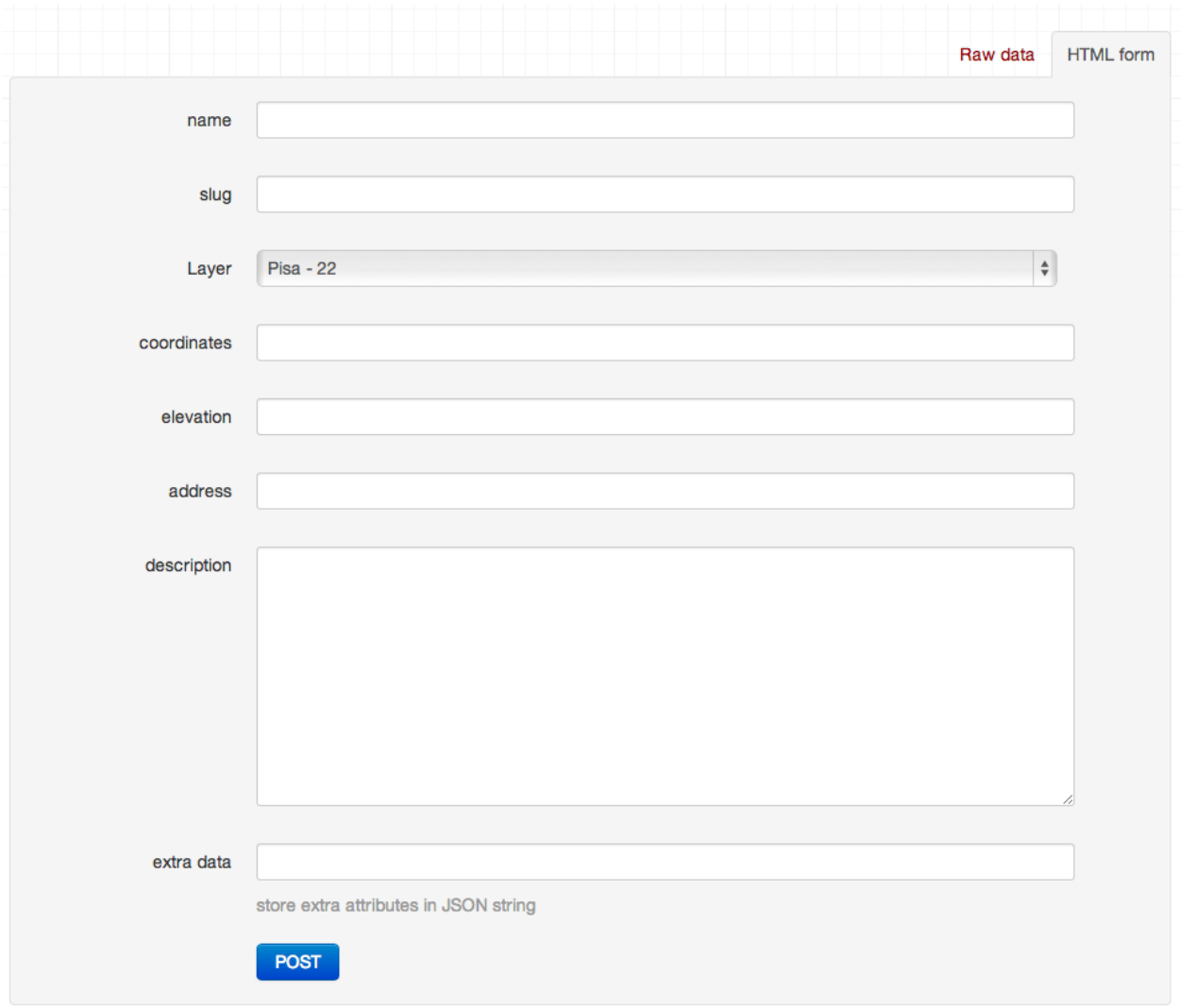
API Documentation

By default when you open the API you will see the **self-documented** HTML version.

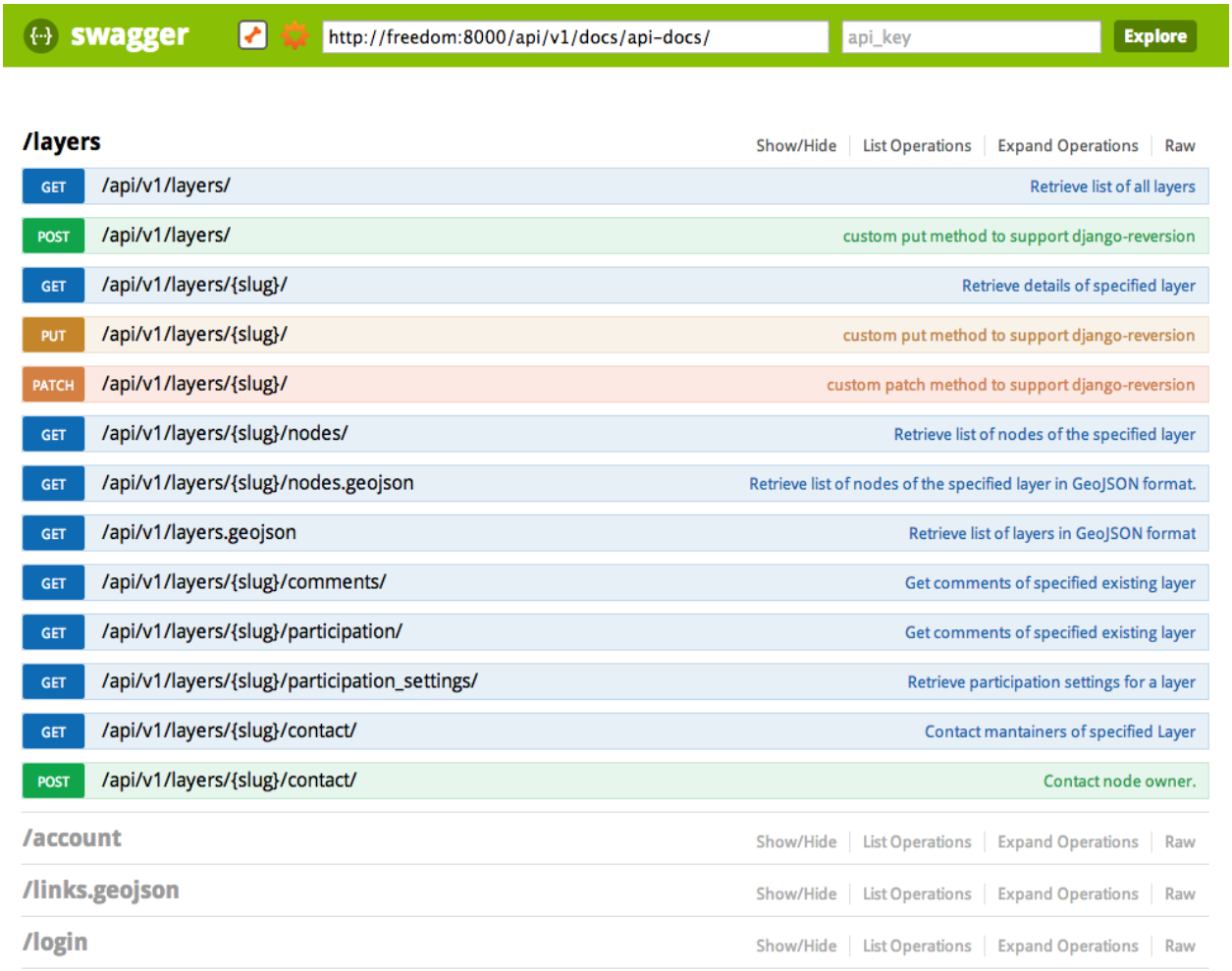


Each resource has a general description of what is its purpose and which operations supports.

The resources which perform write operations will also have an HTML form with which you can experiment and test the API.



There's also another auto generated documentation that makes use of the standard **swagger** format which you can see at **http://localhost:8000/api/v1/docs/**



Nodes

`nodeshot.core.nodes` is the core geographic app of nodeshot.

Nodes have the following features:

- geometry can be a `Point`, a `Polygon` or a `Linestring`
- have to belong to a `Layer` if `nodeshot.core.layers` is enabled (which it is by default)
- have a status, which can be customized in the admin, by default statuses are **potential**, **active**, **planned**
- have other properties like **description**, **address**, ecc.
- can have custom properties by leveraging the `NODESHOT_NODES_HSTORE_SCHEMA` setting

Other modules extend the Node object and add several functionalities like comments, votes, ecc.

Additional features:

- Possibility to customize the default node schema by adding custom fields
- Set the default status of nodes from admin
- Nodes can be published by default or not

- Nodes may store descriptions in HTML format
- Elevation API, a proxy to Google Elevation API, default URL is at `/api/v1/elevation/`

Available settings

`nodeshot.core.nodes` is enabled by default in `nodeshot.conf.settings.INSTALLED_APPS`.

These are the available customizable settings:

- `NODESHOT_NODES_HSTORE_SCHEMA`
- `NODESHOT_NODES_PUBLISHED_DEFAULT`
- `NODESHOT_NODES_REVERSION_ENABLED`
- `NODESHOT_NODES_HTML_DESCRIPTION`
- `NODESHOT_GOOGLE_ELEVATION_API_KEY`
- `NODESHOT_GOOGLE_ELEVATION_DEFAULT_SAMPLING`

NODESHOT_NODES_HSTORE_SCHEMA

default: `None`

`NODESHOT_NODES_HSTORE_SCHEMA`: custom **django-hstore** schema to add new fields on the `Node` model and API.

The following example will add a choice field with a select of 3 choices:

```
# settings.py

NODESHOT_NODES_HSTORE_SCHEMA = [
    {
        'name': 'choice',
        'class': 'CharField',
        'kwargs': {
            'max_length': 128,
            'choices': [
                ('choice1', 'Choice 1'),
                ('choice2', 'Choice 2'),
                ('choice3', 'Choice 3')
            ],
            'default': 'choice1'
        }
    }
]
```

Consult the django-hstore documentation for more information (look for `schema mode`).

NODESHOT_NODES_PUBLISHED_DEFAULT

default: `True`

Whether the default value for the **"is_published"** field on new nodes will be `True` or `False`.

Use `False` if you want new nodes to be reviewed by an admin before showing publicly on the site.

NODESHOT_NODES_REVERSION_ENABLED

default: `True`

Indicates whether the `Node` model can revert changes saved in the history by using django-reversion.

NODESHOT_NODES_HTML_DESCRIPTION

default: `True`

Indicates whether the **"description"** field of the `Node` model allows **HTML** or not.

If `True` an **WYSIWYG** editor will be used in the admin site.

NODESHOT_GOOGLE_ELEVATION_API_KEY

default: `None`

API key of the Google Elevation API.

This setting is optional, but registering an API key is recommended by Google.

See the Google Elevation API documentation for more information.

NODESHOT_GOOGLE_ELEVATION_DEFAULT_SAMPLING

default: `50`

> **Warning:** Setting a very low value may cause the reach of the usage limits of the Google Elevation API.

Calculates automatic sampling to get one point every `x` meters, where `x` is the value specified in `NODESHOT_GOOGLE_ELEVATION_DEFAULT_SAMPLING`.

A bit more explaination is needed: when sending sampled path requests to the Google Elevation API a `samples` parameter is required:

Note: `samples` *(required)* specifies the number of sample points along a path for which to return elevation data. The samples parameter divides the given path into an ordered set of equidistant points along the path.

If no `samples` parameter is specified in the HTTP request to the elevation API resource, nodeshot will take care of it automatically, ensuring there are enough points to represent a meaningful elevation profile.

The default value is **50 meters**, which will return 20 sample points for each kilometer.

Layers

`nodeshot.core.layers` is a django-app that enables nodeshot to group nodes in layers.

A layer may have a geographic area and an organization in charge of it.

The **area** field is required and can be either a **polygon** or a **point**. If a polygon is used, its nodes will have to be contained in it and its center will be calculated automatically; otherwise, if a point is used its nodes will be allowed to be located anywhere and the point will be considered its center.

Available settings

`nodeshot.core.layers` is enabled by default in `nodeshot.conf.settings.INSTALLED_APPS`.

These are the available customizable settings:

- `NODESHOT_LAYERS_HSTORE_SCHEMA`
- `NODESHOT_API_APPS_ENABLED`

NODESHOT_LAYERS_HSTORE_SCHEMA

default: `None`

custom **django-hstore** schema to add new fields on the `Layer` model and API.

The following example will add a category field with a select of 3 choices:

```
# settings.py

NODESHOT_LAYERS_HSTORE_SCHEMA = [
    {
        'name': 'category',
        'class': 'CharField',
        'kwargs': {
            'max_length': 128,
            'choices': [
                ('category1', 'Category 1'),
                ('category2', 'Category 2'),
                ('category3', 'Category 3')
            ],
            'default': 'category1'
        }
    }
]
```

Consult the django-hstore documentation for more information (look for `schema mode`).

NODESHOT_LAYERS_NODES_MINIMUM_DISTANCE

default: `0`

Default value for the field `nodes_minimum_distance` on the `Layer` model.

NODESHOT_LAYERS_REVERSION_ENABLED

default: `True`

Indicates whether the Layer model can revert changes saved in the history by using django-reversion.

NODESHOT_LAYERS_TEXT_HTML

default: `True`

Indicates whether the **"Extended text"** field of the `Layer` model allows **HTML** or not.

If `True` a **WYSIWYG** editor will be used in the admin site.

User Interface

`nodeshot.ui.default` is the default web user interface of nodeshot.

The default interface is replaceable: if you need a radically different web interface you can develop a new one in a separate python package.

Change the first page

By default the first page opened in the UI is a cms page called "home" which you can customize in the admin site under `/admin/cms/page/1/`.

In this page you can put any HTML you want: images, graphs, whatever you think it's most suited to explain what your nodeshot instance does to first time visitors.

If you think you don't need this and you prefer to have a different index page, like for example the map view, just go to `/admin/cms/menuitem/`, and change the order of the pages so that the first page you prefer comes first.

You might also unpublish or delete the links you don't want others to see, as well as add new links to the menu.

Available settings

`nodeshot.ui.default` is enabled by default in `nodeshot.conf.settings.INSTALLED_APPS`.

These are the available customizable settings:

- `LEAFLET_CONFIG`
- `NODESHOT_UI_LEAFLET_OPTIONS`
- `NODESHOT_UI_DISABLE_CLUSTERING_AT_ZOOM`
- `NODESHOT_UI_MAX_CLUSTER_RADIUS`
- `NODESHOT_UI_DATETIME_FORMAT`
- `NODESHOT_UI_DATE_FORMAT`
- `NODESHOT_UI_ADDRESS_SEARCH_TRIGGERS`
- `NODESHOT_UI_LOGO`
- `NODESHOT_UI_VOTING_ENABLED`
- `NODESHOT_UI_RATING_ENABLED`
- `NODESHOT_UI_COMMENTS_ENABLED`
- `NODESHOT_UI_GOOGLE_ANALYTICS_UA`
- `NODESHOT_UI_GOOGLE_ANALYTICS_OPTIONS`
- `NODESHOT_UI_PIWIK_ANALYTICS_BASE_URL`
- `NODESHOT_UI_PIWIK_ANALYTICS_SITE_ID`
- `NODESHOT_UI_PRIVACY_POLICY_LINK`
- `NODESHOT_UI_TERMS_OF_SERVICE_LINK`

LEAFLET_CONFIG

default:

```
{
    'DEFAULT_CENTER': (49.06775, 30.62011),
    'DEFAULT_ZOOM': 4,
    'MIN_ZOOM': 1,
    'MAX_ZOOM': 18,
    'TILES': [
        ('Map', 'https://a.tile.openstreetmap.org/{z}/{x}/{y}.png', '&copy; <a href=
↪"http://www.openstreetmap.org/copyright" target="_blank">OpenStreetMap</a>␣
↪contributors | Tiles Courtesy of <a href="http://www.mapquest.com/" target="_blank">
↪MapQuest</a>  <img src="https://developer.mapquest.com/content/osm/mq_logo.png
↪">'),
        ('Satellite', 'http://server.arcgisonline.com/ArcGIS/rest/services/World_
↪Imagery/MapServer/tile/{z}/{y}/{x}', 'Source: <a href="http://www.esri.com/">Esri</
↪a> &copy; and the GIS User Community ')
    ],
}
```

General options of the map:

- `DEFAULT_CENTER`: default center of the map
- `DEFAULT_ZOOM`: default zoom of the map
- `MIN_ZOOM`: minimum zoom level
- `MAX_ZOOM`: maximum zoom level
- `TILES`: base layers available (eg: map, satellite), learn how to tweak this on the django-leaflet documentation

NODESHOT_UI_LEAFLET_OPTIONS

default:

```
{
    'fillOpacity': 0.7,
    'opacity': 1,
    'dashArray': None,
    'lineCap': None,
    'lineJoin': None,
    'radius': 6,
    'temporaryOpacity': 0.3
}
```

These options control some details of the map:

- `fillOpacity`: fill color opacity of objects on the map
- `opacity`: stroke opacity of objects on the map
- `dashArray`: explained in the Leaflet documentation
- `lineCap`: explained in the Leaflet documentation
- `lineJoin`: explained in the Leaflet documentation
- `radius`: width of the radius circles on the map in pixel, valid only for points (*Nodeshot can display also other shapes*)

- `temporaryOpacity`: when adding a new node the other nodes are dimmed according to this option

Other options like fill color and stroke width are managed in the admin site under `/admin/nodes/status/` because they vary for each status.

NODESHOT_UI_DISABLE_CLUSTERING_AT_ZOOM

default: `12`

At the specified level of zoom clustering of points on the map is disabled.

Setting `1` disables clustering altogether, while setting `0` forces clustering at all zoom levels.

NODESHOT_UI_MAX_CLUSTER_RADIUS

default: `90`

The maximum radius that a cluster will cover from the central marker (in pixels). Decreasing will make smaller clusters.

NODESHOT_UI_DATETIME_FORMAT

default: `dd MMMM yyyy, HH:mm`

`DateTime` formatting according to the jQuery dateFormat docs.

NODESHOT_UI_DATE_FORMAT

default: `dd MMMM yyyy`

`Date` formatting according to the jQuery dateFormat docs.

NODESHOT_UI_ADDRESS_SEARCH_TRIGGERS

default:

```
[
    ',',
    'st.',
    ' street',
    ' square',
    ' road',
    ' avenue',
    ' lane',
    'footpath',
    'via ',
    'viale ',
    'piazza ',
    'strada ',
    'borgo ',
    'contrada ',
    'zona ',
    'fondo ',
    'vico ',
    'sentiero ',
```

```
    'plaza ',
    ' plaza',
    'calle ',
    'carrer ',
    'avenida '
]
```

Special strings that trigger geolocation when searching in the general search bar.

NODESHOT_UI_LOGO

default: `None`

Use this setting to show a custom logo, example:

```
NODESHOT_UI_LOGO = {
    'URL': 'http://yourdomain.com/static/logo.svg',  # value for css rule background-
↪image
    'SIZE': '180px',  # value for css rule background-size
}
```

Note:

- the logo **must be in SVG format**.
- when choosing the size of the logo, mind mobile platforms!

NODESHOT_UI_VOTING_ENABLED

default: `True`

Indicates wheter it is possible to like or dislike nodes.

NODESHOT_UI_RATING_ENABLED

default: `True`

Indicates wheter it is possible to rate nodes (stars).

NODESHOT_UI_COMMENTS_ENABLED

default: `True`

Indicates wheter it is possible to leave comments on nodes.

NODESHOT_UI_CONTACTING_ENABLED

default: `True`

Indicates wheter it is possible to contact other users.

NODESHOT_UI_GOOGLE_ANALYTICS_UA

default: `None`

Google Analytics tracking code.

Example:

```
NODESHOT_UI_GOOGLE_ANALYTICS_UA = 'UA-XXXXXXXX-3'
```

NODESHOT_UI_GOOGLE_ANALYTICS_OPTIONS

default: `auto`

Google Analytics options that will be passed on initialization.

```
NODESHOT_UI_GOOGLE_ANALYTICS_OPTIONS = {
    'cookieDomain': 'none'
}
```

For more information about the options that can be passed see the relative Google Analytics Reference.

NODESHOT_UI_PIWIK_ANALYTICS_BASE_URL

default: `None`

Piwik is a fantastic Open Source Web Analytics tool.

This settings indicates where you installed your own piwik instance.

Example:

```
NODESHOT_UI_PIWIK_ANALYTICS_BASE_URL = 'http://analytics.frm.ninux.org'
```

NODESHOT_UI_PIWIK_ANALYTICS_SITE_ID

default: `None`

Piwik site id.

Example:

```
NODESHOT_UI_PIWIK_ANALYTICS_SITE_ID = 12
```

NODESHOT_UI_PRIVACY_POLICY_LINK

default: `'#/pages/privacy-policy'`

Link to "Privacy Policy" page.

NODESHOT_UI_TERMS_OF_SERVICE_LINK

default: `'#/pages/terms-of-service'`

Link to "Terms of Service" page.

Profiles

`nodeshot.community.profiles` is a django app that adds the following features to the RESTful API:

- user profiles
- user registration
- email confirmation
- email address management (add multiple email addresses)
- login
- logout

Available settings

`nodeshot.community.profiles` is enabled by default in `nodeshot.conf.settings.INSTALLED_APPS`.

These are the available customizable settings:

- `NODESHOT_PROFILES_REGISTRATION_OPEN`
- `NODESHOT_PROFILES_EMAIL_CONFIRMATION`
- `NODESHOT_PROFILES_REQUIRED_FIELDS`

NODESHOT_PROFILES_REGISTRATION_OPEN

default: `True`

Indicates wheter registration is open to the public.

NODESHOT_PROFILES_EMAIL_CONFIRMATION

default: `True`

Indicates wheter new users have to confirm their email.

NODESHOT_PROFILES_REQUIRED_FIELDS

default: `['email']`

Required fields during registration.

Fields added at the moment won't be yet reflected in `nodeshot.ui.default`.

Participation

`nodeshot.community.participation` is a django apps that enables 3 features to make the site more "social":

- **Vote** : like or dislike
- **Comment**: comments on nodes

- **Rating**: 1 to 10 rating

`nodeshot.community.participation` is enabled by default in `nodeshot.conf.settings.INSTALLED_APPS`.

How to disable

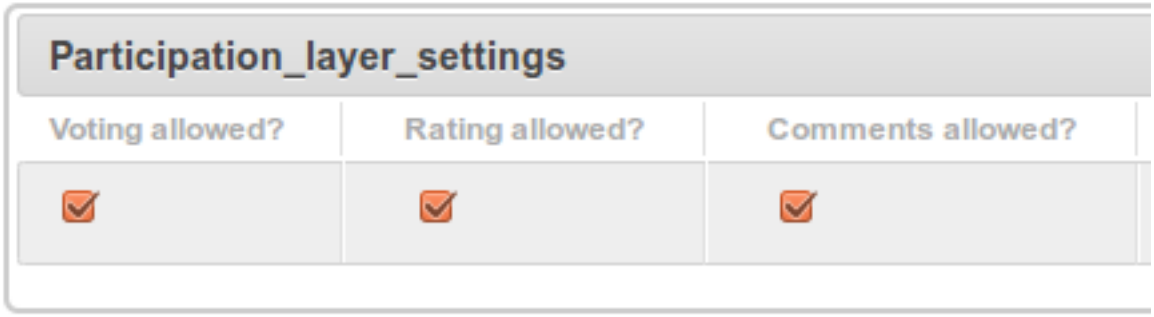
If you need to disable this feature altogether add this at the bottom of your `settings.py`:

```
INSTALLED_APPS.remove('nodeshot.interop.open311')
INSTALLED_APPS.remove('nodeshot.community.participation')
```

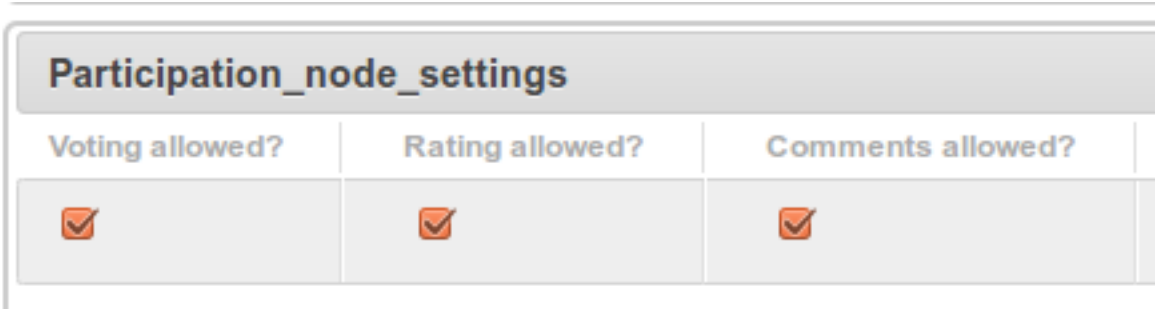
Configuration

Admins can configure which of the above actions can be taken through the Admin Interface.

Participation actions can be enabled or disabled for an entire **layer**:



Or for a single **node**:



API

Inserting data

Comments, votes and ratings can be inserted through *Nodeshot API* or *Open 311 API* .

Querying data

Participation details about nodes are available through *Nodeshot API*.

In particular:

http://<server-name>/api/v1/participation/ will return participation data for all nodes

While:

http://<server-name>/api/v1/<node_slug>/participation/ will return participation data for the specified node

Social Logins

This section describes how to enable social logins for your instance.

Facebook

1. Go to https://developers.facebook.com/
2. Create an app, specify your website domain
3. Grab **App ID** and **App Secret**
4. Fill `FACEBOOK_APP_ID` in `settings.py`
5. Fill `FACEBOOK_API_SECRET` in `settings.py`

Google+

1. Go to https://code.google.com/apis/console#access to request a new API key
2. Create an OAuth app
3. Specify your website URL in `AUTHORIZED JAVASCRIPT ORIGINS` in Google's developer site
4. Specify `URL/complete/google-oauth2/` in `AUTHORIZED REDIRECT URI` in Google's developer site
5. Grab **Client ID** and **Client Secret**
6. Fill `GOOGLE_OAUTH2_CLIENT_ID` in `settings.py`
7. Fill `GOOGLE_OAUTH2_CLIENT_SECRET` in `settings.py`

Github

1. Go to https://github.com/settings/applications
2. Create an app, specify your website domain
3. Grab **Client ID** and **Client Secret**
4. Fill `GITHUB_APP_ID` in `settings.py`
5. Fill `GITHUB_API_SECRET` in `settings.py`

Reload configuration

After any change to `settings.py` you will have to restart your application server:

```
supervisorctl restart uwsgi
```

Mailing

`nodeshot.community.mailing` is a django app that provides mainly 2 features:

- enable users to contact other users and layer managers
- enable admins to send news or emergency communications to all users or a subset of users

`nodeshot.community.mailing` is enabled by default in `nodeshot.conf.settings.INSTALLED_APPS`.

Available settings

These are the available customizable settings:

- `NODESHOT_MAILING_INWARD_REQUIRE_AUTH`
- `NODESHOT_MAILING_INWARD_MAXLENGTH`
- `NODESHOT_MAILING_INWARD_MINLENGTH`
- `NODESHOT_MAILING_INWARD_LOG`
- `NODESHOT_MAILING_OUTWARD_MAXLENGTH`
- `NODESHOT_MAILING_OUTWARD_MINLENGTH`
- `NODESHOT_MAILING_OUTWARD_HTML`
- `NODESHOT_MAILING_OUTWARD_STEP`
- `NODESHOT_MAILING_OUTWARD_DELAY`

NODESHOT_MAILING_INWARD_REQUIRE_AUTH

default: `True`

Whether only authenticated users are allowed to contact other nodes or not.

NODESHOT_MAILING_INWARD_MAXLENGTH

default: `2000`

Maximum length of messages sent by users.

NODESHOT_MAILING_INWARD_MINLENGTH

default: `15`

Minimum length of messages sent by users.

NODESHOT_MAILING_INWARD_LOG

default: `True`

Wether to log messages sent by users in the database or not.

NODESHOT_MAILING_OUTWARD_MAXLENGTH

default: `9999`

Maximum length of messages sent by admins.

NODESHOT_MAILING_OUTWARD_MINLENGTH

default: `50`

Minimum length of messages sent by admins.

NODESHOT_MAILING_OUTWARD_HTML

default: `True`

Allow HTML emails for messages sent by admins; useful for newsletters or similar periodic communications.

Will display a WYSIWYG editor in the amdin.

NODESHOT_MAILING_OUTWARD_STEP

default: `20`

Emails won't be sent all in one go, the sending will be divided in several steps.

This setting configures how many emails to send before pausing for the number of seconds set in `NODESHOT_MAILING_OUTWARD_DELAY`.

NODESHOT_MAILING_OUTWARD_DELAY

default: `10`

Number of seconds to wait after one *step* (as explained in `NODESHOT_MAILING_OUTWARD_STEP`) is completed.

Debugging emails in the shell

If you need to test email sending in a development/test environment, you can use a debug SMTP server which will print out the outgoing emails on your terminal shell.

First of all, ensure `DEBUG = True` in your `settings.py`.

Then run the debug SMTP server with:

```
python -m smtpd -n -c DebuggingServer localhost:1025
```

When in `DEBUG` is `True`, Nodeshot will send emails to the port 1025 by default.

The email contents and log data will be printed out on your terminal shell, this way you'll be able to test email related features without sending any real email.

Web Sockets

Web Sockets in nodeshot are implemented through an experimental django app in `nodeshot.core.websockets`.

This app will be removed in favour of a better functioning module soon.

How to disable

If you need to disable this feature altogether add this at the bottom of your `settings.py`:

```
INSTALLED_APPS.remove('nodeshot.core.websockets')
```

Synchronizers

`nodeshot.interop.sync` is a django-app that enables nodeshot to build an abstraction layer between itself and other third party web-applications which deal with georeferenced data.

There are mainly four strategies through which we can achieve interoperability with third party web apps:

- **periodic synchronization**: data is imported periodically into the local database by a background job which reads an external source
- **event driven synchronization**: data is exported to a third party API whenever local data is added, changed or deleted
- **RESTful translator**: data is retrieved on the fly and converted to json/geojson, no data is saved in the database
- **mixed**: custom synchronizers might implement mixed strategies

These strategies are implemented through **"Synchronizers"**.

New synchronizers can be written ad-hoc for each application that need to be supported.

Internal dependencies

To enable this feature, the following apps must be listed in `settings.INSTALLED_APPS`:

- nodeshot.core.layers
- nodeshot.core.nodes
- nodeshot.interop.sync

By default these three apps are installed.

Required settings

The module `nodeshot.interop.sync` is activated by default in `nodeshot.conf.settings.INSTALLED_APPS`.

For periodic synchronization `CELERYBEAT_SCHEDULE` must be uncommented in your `settings.py`:

```
from datetime import timedelta

CELERYBEAT_SCHEDULE.update({
    'synchronize': {
        'task': 'nodeshot.interop.sync.tasks.synchronize_external_layers',
        'schedule': timedelta(hours=12),
    },
    # ... other tasks ...
})
```

You might want to tweak how often data is synchronized, in the previous example we configured the task to run every 12 hours.

Layer configuration

Interoperability is configured at layer level in the admin interface.

A layer must be flagged as **"external"**, after doing so a new box labeled "External layer info" will appear in the bottom of the page. If the layer is new and not saved in the database proceed to save and reload the admin page.

Then change the synchronizer field and the configuration fields will appear.

Each synchronizer has different fields, an brief explaination of the default synchronizers follows.

Nodeshot (RESTful translator)

This synchronizer is a **RESTful translator** and allows to reference the nodes of an external nodeshot instance.

There are two required configuration keys:

- **layer url**: URL of the layer API resource, eg: `https://test.map.ninux.org/api/v1/layers/rome/`
- **verify ssl**: indicates wether the SSL certificate of the external layer should be verified or not; if checked self signed certificates won't work

There is no periodic synchronization needed because this synchronizer grabs the data on the fly.

GeoJSON (periodic sync)

This synchronizer implements the **periodic synchronization** strategy and therefore needs to be enabled in the `CELERYBEAT_SCHEDULE` setting.

The main configuration keys are:

- **url**: URL to retrieve the geojson file
- **verify_ssl**: indicates wether the SSL certificate of the external layer should be verified or not; if checked self signed certificates won't work
- **default status**: status to be used for new nodes, to use the system default leave blank

There are other configuration keys which enable to parse geojson files which use radically different names for corresponding fields.

- **name**: corresponding name field, for example, on the data source file the name field could be labeled **title**
- **status**: corresponding status field, if present

- **description**: corresponding description field, if present
- **address**: corresponding address field, if present
- **is_published**: corresponding is_published field, if present
- **user**: corresponding user field, if present
- **elev**: corresponding elev field, if present
- **notes**: corresponding notes field, if present
- **added**: corresponding added field, if present
- **updated**: corresponding updated field, if present

GeoRSS (periodic sync)

This synchronizer implements the **periodic synchronization** strategy and therefore needs to be enabled in the `CELERYBEAT_SCHEDULE` setting.

The main configuration keys are:

- **url**: URL to retrieve the georss file
- **verify_ssl**: indicates wether the SSL certificate of the external layer should be verified or not; if checked self signed certificates won't work
- **default status**: status to be used for new nodes, to use the system default leave blank

There are other configuration keys which enable to parse georss files which use radically different names for corresponding fields.

- **name**: corresponding name field, defaults to **title**
- **status**: corresponding status field, if present
- **description**: corresponding description field, if present
- **address**: corresponding address field, if present
- **is_published**: corresponding is_published field, if present
- **user**: corresponding user field, if present
- **elev**: corresponding elev field, if present
- **notes**: corresponding notes field, if present
- **added**: corresponding added field, defaults to **pubDate**
- **updated**: corresponding updated field, if present

OpenWisp (periodic sync)

This synchronizer inherits from the **GeoRSS** synchronizer, the available options and configurations are the same.

The only difference is that this synchronizer is designed to grab data from the GeoRSS file produced by OpenWISP Geographic Monitoring.

Sync management command

This is the command which is used to perform **periodic synchronization**, use `--help` to know its options:

```
python manage.py sync --help
```

Sync a specific layer:

```
python manage.py sync layer-slug
```

Sync multiple layers by specifying space separated layer slugs:

```
python manage.py sync layer1-slug layer2-slug
```

Sync all layers is as simple as:

```
python manage.py sync
```

Sync all layers except those specified in –exclude:

```
python manage.py sync --exclude=layer1-slug,layer2-slug

# spaces are allowed as long as string is wrapped in quotes/doublequotes

python manage.py sync --exclude="layer1-slug, layer2-slug"
```

Writing new synchronizers

To write new synchronizers, you should extend the class `GenericGisSynchronizer` in `/nodeshot/interoperability/synchronizers/base.py`:

```
# my_very_cool_app.py

from nodeshot.interop.sync.synchronizer.base import GenericGisSynchronizer

class MyVeryCoolApp(GenericGisSynchronizer):
    """ Synchronizer for my MyVeryCoolApp """
    pass
```

Note: this section is a work in progress.

Once the file is saved and you are sure it's on your pythonpath you have to add a tuple in `settings.NODESHOT_SYNCHRONIZERS` in which the first element is the path to the file and the second element is the name you want to show in the admin interface in the *"Synchronizer"* select:

```
NODESHOT_SYNCHRONIZERS = [
    ('myproject.synchronizers.my_very_cool_app.MyVeryCoolApp', 'MyVeryCoolApp'),
]
```

This will add your new synchronizer to the default list.

Third party packages

- nodeshot-citysdk-synchronizers: https://github.com/nemesisdesign/nodeshot-citysdk-synchronizers

Open311 API

Nodeshot comes with a self-documented open311 API, in order to insert nodes, comments, votes or ratings as service requests, according to the Open 311 specification (http://open311.org/).

This app depends on the **participation** app, be sure to read its *documentation*.

Settings

`nodeshot.interop.open311` and its dependencies are enabled by deafult.

In case you want to disable this app consult the **"Uninstall"** section below.

The available settings for the Open311 app are the following:

- `NODESHOT_OPEN311_DISCOVERY`
- `NODESHOT_OPEN311_METADATA`
- `NODESHOT_OPEN311_TYPE`
- `NODESHOT_OPEN311_STATUS`

`NODESHOT_OPEN311_DISCOVERY` is a dictionary containing service discovery metadata. Inside it, you can define different endpoints (e.g production, test, development, ecc..)

See http://wiki.open311.org/Service_Discovery for more details.

`NODESHOT_OPEN311_METADATA` and `NODESHOT_OPEN311_TYPE` need to be changed only in order to completely redefine the implementation of Nodeshot Open 311 service definition.

See http://wiki.open311.org/GeoReport_v2 for details but you probably don't want to do this!

`NODESHOT_OPEN311_STATUS` is a dictionary, containing the values that have been inserted in 'Status' model as keys, and 'open' or 'closed' as possible values. It is important that the keys of this dictionary match exactly the values of the slug fields contained in the Status model records, otherwise the application will either throw an exception (if in DEBUG mode) or defaults to "closed" (production).

In its simpliest form, the configuration would be this:

```
'STATUS' : {
    'open' : 'open',
    'closed' : 'closed',
}
```

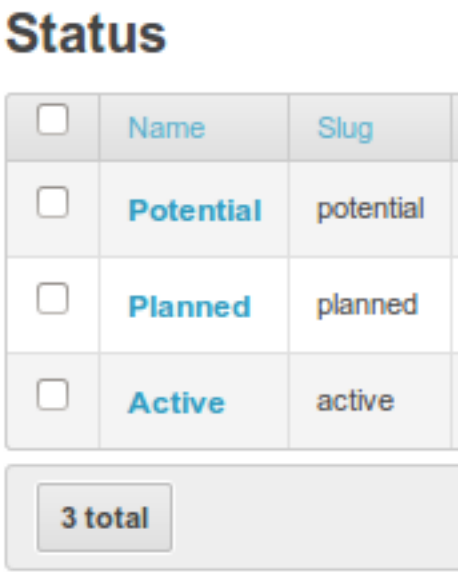
Or, if more statuses are possible in your configuration, like in the example below:

each status can be mapped to one of the two values 'open' or 'closed', depending on your needs:

```
'STATUS' : {
    'potential' : 'open',
    'planned' : 'open',
    'active' : 'closed'
}
```

Uninstall

To uninstall `nodeshot.interop.open311` simply remove it from your `settings.INSTALLED_APPS`




```
# setings.py

# import the default nodeshot settings
# do not move this import
from nodeshot.conf.settings import *

# ------ All settings customizations must go here ------ #

INSTALLED_APPS.remove('nodeshot.interop.open311')
```

Import data from older versions

The new version provides an integrated tool that allows to import data from older nodeshot versions (0.9.x).

> **Warning:** this tool is a work in progress, please report any bug in our Mailing List.

Internal dependencies

For the **oldimporter** module to work, the following apps must be listed in `settings.INSTALLED_APPS`:

- nodeshot.core.nodes
- nodeshot.core.layers
- nodeshot.networking.net
- nodeshot.networking.links
- nodeshot.community.mailing
- nodeshot.community.profiles

By default these apps are included in `nodeshot.conf.settings` so you won't need to do anything.

Install database drivers

Most production installations of old nodeshot versions used MySQL (development quick install were done with SQlite).

Because these drivers are not installed by default with the default install procedure, you will have to install them now.

For MySQL you can do:

```
sudo apt-get install libmysqlclient-dev

workon nodeshot  # activate virtualenv
pip install MySQL-python
```

Enable in settings.py

Uncomment the following section in `settings.py` and tweak the settings `ENGINE`, `NAME`, `USER`, `PASSWORD`, `HOST` and `PORT` according to your configuration:

```
# Import data from older versions
# More info about this feature here: http://nodeshot.readthedocs.org/en/latest/topics/
↪oldimporter.html
#'old_nodeshot': {
#     'ENGINE': 'django.db.backends.mysql',
#     'NAME': 'nodeshot',
#     'USER': 'user',
#     'PASSWORD': 'password',
#     'OPTIONS': {
#             "init_command": "SET storage_engine=INNODB",
#     },
#     'HOST': '',
#     'PORT': '',
#}
```

And set `NODESHOT_OLDIMPORTER_DEFAULT_LAYER` (object id/primary key) to your default layer:

```
NODESHOT_OLDIMPORTER_DEFAULT_LAYER = <id>
```

Replace `<id>` with the id of your default layer.

If you followed exactly the instructions in this document you can leave the default `NODESHOT_OLDIMPORTER_STATUS_MAPPING` setting unchanged.

Preparation

Due to some major differences between the old and the new version some manual preparation needs to be done.

1. Ensure your old database is reachable

In order for the **oldimporter** to work it musts be able to connect to the remote old database.

If your old database is MySQL or PostgreSQL you should tweak its configuration to allow connections from the IP/hostname where the new version of nodeshot is installed.

If your old database is Sqlite you can just copy the file to the new machine.

2. Create Layers

Then ensure you have some layers defined in your admin interface. Open the browser and go to **/admin/layers/layer** (or follow the links from the admin index), if you see any layer defined, you are ready to proceed, if not you should create one or more layers.

If you **specify the area of each layer**, the importer will be able to insert the old nodes into the right layer. It's a good thing to do it!

If you don't want to lose any node, you should create a **default layer** in which the script will automatically put all those old nodes which have coordinates that are not comprised in any of your newly created layers.

If no default layer is specified the nodes which have coordinates not comprised in any layer will be discarded.

Import data

> **Warning:** The first import should start with a clean database

First of all, enable your python-virtualenv if you haven't already:

```
workon nodeshot
```

Ready? Go!:

```
python manage.py import_old_nodeshot
```

If you want to see what the importer is doing behind the scenes raise the verbosity level:

```
python manage.py import_old_nodeshot --verbosity=2
```

If you want to save the output for later inspection try this:

```
python manage.py import_old_nodeshot --verbosity=2 | tee import_result.txt
```

Wait for the importer to import your data, when it finishes it will ask you if you are satisfied with the results or not, if you enter "No" the importer will delete all the imported records.

If the importer runs into an uncaught exception it will automatically delete all the imported data.

If you get such an error notify us and we'll try to fix it.

In case you don't want the importer data to be deleted you can use the `--nodelete` option.

Command options

- `--verbosity`: verbosity level, can be 0 (no output), 1 (default), 2 (verbose), 3 (very verbose)
- `--noinput`: suppress all user prompts
- `--nodelete`: do not delete imported data in case of errors

Periodic sync

You can run the importer periodically and it will try to import new data.

This process can be handy while you test the new version but before you launch your service to your audience we advise to reset everything and run the importer again on a clean database.

It is better to specify the `--nodelete` option in order to avoid automatic deletion of data in case of errros:

```
python manage.py import_old_nodeshot --nodelete
```

To automate the periodic import add the following dictionary in your `CELERYBEAT_SCHEDULE` setting:

```
CELERYBEAT_SCHEDULE = {

    # ...

    'import_old_nodeshot': {
        'task': 'nodeshot.interop.oldimporter.tasks.import_old_nodeshot',
        'schedule': timedelta(hours=12),
        # pass --noinput and --nodelete options
        'kwargs': { 'noinput': True, 'nodelete': True }
    },

    # ...

}
```

This assumes that celery and celerybeat are configured and running correctly.

Deactivate oldimporter

When you are finished using the oldimporter module you can disable it by commenting the `DATABASES['old_nodeshot']` setting.

How does the importer work?

Let's explain some technical details, the flow can be divided in 7 steps.

1. Retrieve all nodes

The first thing the script will do is to retrieve all the nodes from the old database and convert the queryset in a python list that will be used in the next steps.

2. Extract user data from nodes

Since in old nodeshot there are no users but each node contains data such as name, email, and stuff like that, the script will create user accounts:

- loop over nodes and extract a list of unique emails
- each unique email will be a new user in the new database
- each new user will have a random password set
- save users, email addresses

3. Import nodes

- **USER**: assign owner (the link is the email)
- **LAYER: assign layer (layers must be created by hand first!):**
    1. if node has coordinates comprised in a specified layer choose that
    2. if node has coordinates comprised in more than one layer prompt the user which one to choose
    3. **if node does not have coordinates comprised in any layer:**
        (a) use default layer if specified (configured in settings)
        (b) discard the node if no default layer specified
- **STATUS: assign status depending on configuration:** `settings.NODESHOT_OLDIMPORTER_STATUS_MAPPING` must be a dictionary in which the key is the old status value while the value is the new status value if `settings.NODESHOT_OLDIMPORTER_STATUS_MAPPING` is False the default status will be used
- **HOSTPOT**: if status is hotspot or active and hotspot add this info in the *HSTORE* data field

4. Import devices

In this step the script will import devices and create any missing routing protocol.

5. Import interfaces, ip addresses, vaps

In this step the script will import all interfaces, ip addresses and other detailed device info.

6. Import links

In this step the script will import all the available links unless `settings.NODESHOT_OLDIMPORTER_IMPORT_LINKS` is set to `False`.

7. Import Contacts

In this step the script will import the contact logs.

Sentry integration

Nodeshot can be monitored by sentry pretty easily.

All you have to do is to set your sentry API key (also known as DSN URL) in your `settings.py`. Look for `RAVEN_CONFIG`:

```
# settings.py

# sentry integration
RAVEN_CONFIG = {
    'dsn': 'https://<api-public-key>:<api-secret-key>@<sentry.host>/<id>?timeout=5&
↪verify_ssl=0',
}
```

After this change remember to reload the **uwsgi** application server:

```
supervisorctl restart uwsgi
```

Connectors

Functionality moved in a separate project: NetEngine.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search