DNStorm Documentation

Release 1.0

Vinicius Massuchetto

August 05, 2015

CHAPTER 1

About

DNStorm is an experiment in decision-making theory made by Vinicius Massuchetto and Willy Hoppe de Sousa for the Master Program in Nuclear Technology Applications of the Institute of Energy and Nuclear Research and University of São Paulo in Brazil.

This is a simple collaborative platform that allows managers to state problems and ask for contributions of quantified ideas from a web brainstorming processes that will build the problem and solution presentation in the format of a strategy table.

The project's fancy page presents the software in a non-technical language. Sphinx documentation can be found at Read the Docs. A live demo and experimental environment can be found on Heroku.

CHAPTER 2

Status

The development status is in its alpha stages. Feel free to help by reporting bugs and development suggestions on Github issues.

CHAPTER 3

Build a test and development installation

The project uses Python 2.7. Make sure your *python*, *virtualenv* and *pip* binaries meets this version.

Clone the repository and go the project's root to build the environment:

```
git clone git@github.com:vmassuchetto/dnstorm.git
cd dnstorm
```

Start a virtual environment, load it and install the required packages from the `requirements.txt` file. After this, make sure all the command line used from here is executed in this virtual environment (has the `(env)` on the command prompt).

```
virtualenv --distribute env
source env/bin/activate
pip install -r requirements.txt
```

Setup the SQLite3 database:

```
python manage.py syncdb
python manage.py migrate
```

Run your server:

```
python manage.py runserver
```

The application might be running at `http://localhost:8000`.

CSS

For hosting environment reasons, the compiled `static/scss/app.css` is already in the project's repository. That means you don't need to go further if you're not developing.

The project's CSS uses the *Foundation <http://foundation.zurb.com>_* framework and is generated with Sass. DNStorm uses a set of Grunt and Bower packages for the static files. To install everything via *nodejs*:

```
npm install
./node_modules/bower/bin/bower install
```

And to generate the static CSS:

```
./node_modules/grunt-cli/bin/grunt build
```

If you're editing the main `static/scss/app.scss` file, you might want to use `grunt watch` instead.

CHAPTER 5

E-mails

E-mail receival in development mode can be checked by a SMTP debugging server:

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

CHAPTER 6

Localization

The PO and MO files for each language are located in `dnstorm/app/locale/<locale_code>/LC_MESSAGES`. To generate a PO file for a given locale code run this:

```
source env/bin/activate
cd dnstorm/app/
python ../../manage.py makemessages -l <locale code>
```

Documentation

To generate the Sphinx documentation files:

```
source env/bin/activate
cd docs
make <documentation format>
```

Usually you might want to replace `<documentation format>` with `html`.

CHAPTER 8

Deploying on Heroku

In order to successfully deploy on Heroku this project needs the following setup:

- `package.json` file must be deleted
- `bower.json` must be deleted
- `dnstorm/app/static/components` directory must be included
- `dnstorm/settings/heroku.py` file must be created accordingly to the sample configuration file on `dnstorm/settings/heroku-sample.py`

You can create another `heroku` branch to deploy to the `heroku` remote like this:

```
git push heroku heroku:master
```

Models

class `dnstorm.app.models.`**`Alternative`**`(`**args*, ***kwargs*`)`
Alternatives are the strategy table rows where ideas can be allocated.

`fill_data``(`*user=False*`)`
Fill the alternative with problem and user-specific data.

class `dnstorm.app.models.`**`Comment`**`(`**args*, ***kwargs*`)`
Comments that can be made for ideas or problems.

class `dnstorm.app.models.`**`Criteria`**`(`**args*, ***kwargs*`)`
When creating a problem, managers should define some criterias as a reference for the ideas submitted by users. These will also be the columns for the strategy table of the problem.

class `dnstorm.app.models.`**`Idea`**`(`**args*, ***kwargs*`)`
Ideas are the second main entity in the platform, as the problem-solving process requires idea generation and participation of users. These will after compose the strategy table.

`fill_data``(`*user=False*`)`
Fill the idea with problem and user-specific data.

class `dnstorm.app.models.`**`IdeaCriteria`**`(`**args*, ***kwargs*`)`
Builds the relationship between ideas and criteria for the user to enter a description about the judgements of each criteria of the problem.

class `dnstorm.app.models.`**`Invitation`**`(`**args*, ***kwargs*`)`
Invitations are used to add non-registered users as contributors of problems. The user will have access granted to the problem on registration.

class `dnstorm.app.models.`**`Option`**`(`**args*, ***kwargs*`)`
Meta-based table to store general site options retrieved via the `get` method.

Attributes:

- `name` unique entry key
- `value` value for the key

`get``(`**args*`)`
The site options are defined and saved by the OptionsForm fields, and this method ensures that some value or a default value will be returned when querying for an option value. *None* is returned if the option name is invalid.

`get_all``()`
Get all the default values.

get_defaults(**args*, ***kwargs*)
These are some default values that are used in templates and somewhere else. They are supposed to be overwritten by values on database.

update(**args*)
Update a value based on the option name.

class dnstorm.app.models.**Problem**(**args*, ***kwargs*)
Problems are the central entity of the platform, as everything goes around them. This is no more than the suubject of discussion for generating ideas and a strategy table.

Permissions flags are the following:

- published: if the problem is published or in draft mode
- open: open contribution mode – anyone will be able to edit objects
- public: if the problem can be viewed by non-collaborators

Attributes:

- `last_activity` Gets updated in favor of the `ActivityManager` ordering every time an idea or a comment is made for this problem.

class dnstorm.app.models.**Vote**(**args*, ***kwargs*)
Votes for ideas, comments or alternatives.

CHAPTER 10

Utility functions

`dnstorm.app.utils.`**`activity_count`**`(`*obj*`)`
Increments the activity stream counter of the followers of the given object to make a Facebook look and feel on the top bar.

`dnstorm.app.utils.`**`activity_register`**`(`*_user*, *_action_object*`)`
Registers and activity when an object is saved. Takes a diff with a previous version of edited objects, put it as message content in a timeline, uses the verbs 'created' or 'edited' for actions on actstream.

`dnstorm.app.utils.`**`activity_reset_counter`**`(`*user*`)`
Resets the activity stream counter for a user.

`dnstorm.app.utils.`**`email_context`**`(`*more_context={}*`)`
Puts `more_context` with the standard context variables required for sending e-mails.

`dnstorm.app.utils.`**`get_object_or_none`**`(`*klass*, **args*, ***kwargs*`)`
Get an object and return None if not found.

`dnstorm.app.utils.`**`get_option`**`(`*name*`)`
Wrapper for `get` method of Option class. A tribute to WordPress.

`dnstorm.app.utils.`**`get_user`**`(`*username*`)`
Return user information.

`dnstorm.app.utils.`**`is_email`**`(`*_string*`)`
Checks if a string is an e-mail.

`dnstorm.app.utils.`**`update_option`**`(`*name*, *value*`)`
Wrapper for `update` method of Option class. A tribute to WordPress.

CHAPTER 11

Ajax actions views

CHAPTER 12

Base views

Signals

`dnstorm.app.signals.`**`create_notice_types`**(*app*, *created_models*, *verbosity*, ***kwargs*)

Register notification types for django-notification.

`dnstorm.app.signals.`**`login_on_registration`**(*sender*, *user*, *request*, ***kwargs*)

Logs in the user after registration.

CHAPTER 14

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index