Django Flag App

Release v1.0.0

Nov 17, 2020

Contents:

1 django-flag-app **1**
- 1.1 Installation . . . 1
- 1.2 Usage . . . 2
- 1.3 Contributing . . . 3

2 Web API **5**
- 2.1 Setup . . . 5
- 2.2 API Actions . . . 5

3 Settings **7**
- 3.1 FLAG_ALLOWED . . . 7
- 3.2 FLAG_REASONS . . . 7

4 Style Customization **9**
- 4.1 Overriding templates . . . 9

5 Contributing to Django Flag App **11**
- 5.1 Development . . . 11
- 5.2 Testing . . . 11

6 Changelog **13**
- 6.1 1.0.0 . . . 13
- 6.2 0.1.1 . . . 13
- 6.3 0.1.0 . . . 13

7 Indices and tables **15**

CHAPTER 1

django-flag-app

A pluggable django application that adds the ability for users to flag(or report) your models.

1.1 Installation

Install using `pip`

```
$ pip install django-flag-app
```

If you want, you may install it from the source, grab the source code and run `setup.py`.

```
$ git clone git://github.com/abhiabhi94/django-flag-app.git
$ cd django-flag-app
$ python setup.py install
```

1.2 Usage

1.2.1 Add app

To enable `django_flag_app` in your project you need to add it to `INSTALLED_APPS` in your projects `settings.py` file:

```
INSTALLED_APPS = (
    ...
    'flag',
    ...
)
```

1.2.2 Add URL

In your root `urls.py`:

```
urlpatterns = patterns(
        path('admin/', admin.site.urls),
        path('flag/', include('flag.urls')),
        ...
        path('api/', include('flag.api.urls')),  # only required for API Framework
        ...
    )
```

1.2.3 Migrate

Run the migrations to add the new models to your database:

```
python manage.py migrate flag
```

1.2.4 Connect the flag model with the target model

In `models.py` add the field **flags** as a `GenericRelation` field to the required model.

E.g. for a `Post` model, you may add the field as shown below:

```
from django.contrib.contenttypes.fields import GenericRelation

from flag.models import Flag


class Post(models.Model):
    user = models.ForeignKey(User)
    title = models.CharField(max_length=200)
    body = models.TextField()
    # the field name should be flags
    flags = GenericRelation(Flag)
```

1.2.5 Use template tag

If you want to use web API, this step is not required. See further instructions at *API Actions*.

`render_flag_form` tag requires 2 required positional arguments:

1. Instance of the targeted model.
2. User object.

To render the `flag` form for a the instance `post`, place this inside your detail view, perhaps in some template of the sort `postdetail.html`.

```
{% render_flag_form post user %}
```

1.3 Contributing

Please see the instructions at *Contributing to Django Flag App*.

CHAPTER 2

Web API

django-flag-app uses django-rest-framework to expose a Web API that provides developers with access to the same functionality offered through the web user interface.

The available actions with permitted user are as follows:

- Flag content. (authenticated users)
- Unflag content. (user who has previously flagged that content)

2.1 Setup

To integrate the flag API into your app, just follow the instructions as mentioned *Usage*.

2.2 API Actions

All actions can only be performed by authenticated users. Authorization must be provided as a TOKEN or USERNAME:PASSWORD. `POST` is the allowed method for all requests.

All available actions are explained below:

2.2.1 Flag content

This action can be performed by providing the URL with data queries related to the content type.

The request requires the following parameters:

- `model_name`: is the model name of the content type that have flags associated with it.
- `model_id`: is the id of an object of that model
- `app_name`: is the name of the app that contains the model.

- `reason`: number corresponding to the reason(e.g. 1, 2, 3).
- `info`: '' (This is only required if the reason is `100` (`Something else`))

For example, to flag a content of second object (id=1) of a model (content type) called `post` inside the app(django app) `post`. You may do the following:

```
$ curl -X POST -u USERNAME:PASSWORD -H "Content-Type: application/json" -d "{'app_name
↪': 'post','model_name': 'post', 'model_id': 1,'reason': 1,'info': ''}" http://
↪localhost:8000/api/flag/
```

2.2.2 Un-Flag Content

To un-flag a **FLAGGED** content, set reason value to `0` or remove it from the request.

```
$ curl -X POST -u USERNAME:PASSWORD -H "Content-Type: application/json" -d "{'app_name
↪': 'post','model_name': 'post', 'model_id': 1}" http://localhost:8000/api/flag/
```

CHAPTER 3

Settings

django-flag-app has a few configuration options that allow you to customize it.

3.1 FLAG_ALLOWED

The number of flags allowed before a content is set as flagged. Defaults to `10`.

3.2 FLAG_REASONS

The reasons for which a content can be flagged. Users will have a choose one of these before they flag a content. This a list of tuples. Defaults to:

```
from django.utils.translation import gettext_lazy as _

[
    (1, _('Spam | Exists only to promote a service')),
    (2, _('Abusive | Intended at promoting hatred')),
]
```

Remember that `(100, _('Something else')` will always be appended to this list.

CHAPTER 4

Style Customization

The flag app has been built in a way that you can customise its look and feel completely. Most of the styles can be customised through HTML classes.

In case, you feel there is some customisation that can be added, feel free to open an issue.

The template structure of the flag app looks something like this:

```
├── templates
└── flag
    ├── flag_form.html
    └── flag_icon.html
```

4.1 Overriding templates

To customise a template,

- Create `flag` folder inside templates directory.
- Inside it, create a new template file, giving it the same name as that of the default template that needs to be overridden.

For example, to override the HTML classes of `submit button`

create `templates/flag/flag_form.html` (assuming all your templates are placed under the directory `templates`)

```
{% extends "flag/flag_form.html" %}

{% block cls_flag_modal_submit %}
my-class
{% endblock cls_flag_modal_submit %}
```

4.1.1 Blocks

Please refer to this table when using blocks to customise HTML classes

Block	Use	HTML element
`{% block cls_flag %}`	complete flag element	`div`
`{% block cls_flag_icon_img %}`	flag icon image element	`div`
`{% block cls_flag_modal %}`	modal that appears when flagging	`div`
`{% block cls_flag_modal_content %}`	modal content	`div`
`{% block cls_flag_modal_close %}`	the cross icon(close button) inside the modal	`span`
`{% block cls_flag_modal_form_div %}`	the `div` element containing the modal form	`div`
`{% block cls_flag_modal_form %}`	modal form element	`form`
`{% block cls_flag_modal_title %}`	the text containing modal title	`div`
`{% block cls_flag_modal_reasons %}`	the element that displays reasons for flagging	`tr`
`{% block cls_flag_modal_reason %}`	individual reasons	`input`
`{% block cls_flag_modal_info %}`	text box which appears when `Somthing else` is selected	`textarea`
`{% block cls_flag_modal_submit %}`	submit button inside the modal	`input`

4.1.2 Flag Icon

To change the flag icon, just override the template `flag_icon.html` as explained above. Make sure that you add the property `class="flag-icon {% if has_flagged %}user-has-flagged{% else %}user-has-not-flagged{% endif %}"` to your HTML element. These classes are used by `javascript` files.

For other customisation, please refer to the *Blocks* above

CHAPTER 5

Contributing to Django Flag App

There are many ways to contribute to the project. You may improve the documentation, address a bug, add some feature to the code or do something else. All sort of contributions are welcome.

5.1 Development

To start development on this project, fork this repository and follow the following instructions.

```
# clone the forked repository
$ git clone YOUR_FORKED_REPO_URL

# create a virtual environment
$ python3 -m venv venv
# activate the virtual environment
$ source venv/bin/activate
# install dependencies
(venv) $ pip install -e . -r testapp/requirements.txt
# migrate the migrations to the database
(venv) $ python manage.py migrate
# create data
(venv) $ python manage.py create_initial_data
# start the development server
(venv) $ python manage.py runserver
```

5.2 Testing

To run tests against a particular `python` and `django` version installed inside your virtual environment, you may use:

```
(venv) $ python manage.py test
```

To run tests against all supported `python` and `django` versions, you may run:

```
# install dependency
(venv) $ pip install tox
# run tests
(venv) $ tox
```

CHAPTER 6

Changelog

6.1 1.0.0

- Fix API response.
- Change parameter name to set amount of reports/flags to `FLAG_ALLOWED`.

6.2 0.1.1

- Fix a bug that prevented icon's title when flagging/unflagging using templates.
- Fix template to redirect unauthenticated users to login page when try to flag.
- Fix links in the readme on PyPi.

6.3 0.1.0

First release

CHAPTER 7

Indices and tables

- genindex
- modindex
- search