django-flexible-plans Documentation

Release 0.1.0

Tobia Ghiraldini

May 02, 2019

Contents

Contents:

CHAPTER 1

django-flexible-plans

Independent and reusable Plan, Subscription app used to build the Django Subscription Plan System

Django-Subscription-Plan-System

1.1 Status

Under heavy development. Not ready for use yet. Please feel free to join and contribute.

1.2 Documentation

The full documentation is at https://django-flexible-plans.readthedocs.io.

1.3 Quickstart

Install django-flexible-plans:

```
pip install django-flexible-plans
```

Add it to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    ...
    'flexible_plans.apps.FlexiblePlansConfig',
```

(continues on next page)

(continued from previous page)

```
    ...
)
```

Add django-flexible-plans's URL patterns:

```
from flexible_plans import urls as flexible_plans_urls


urlpatterns = [
    ...
    url(r'^', include(flexible_plans_urls)),
    ...
]
```

Create Features PlanFeatures, Plans through the admin panel to let users activate their subscriptions.

1.4 Features

Plans have Features that describe them and what their subscribers can do or their quotas Users can subscribe to Plans and their Subscription can be activated, deactivated, renewed, prorated, upgraded or downgraded.

1.5 Roadmap

- Configurable, swappable Plan Model, to customize the Plan Class behaviour
- Dynamic imports from other app in the ecosystem to handle all the other aspects of having a plan subscription system
- Multiple plan types, with support of any combination of quotas, features, permissions.
- Multiple plan/subscription provider support

1.6 Running Tests

Does the code actually work?

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install tox
(myenv) $ tox
```

1.7 Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage
- django-model-utils
- django-swappable-models

- django-countries

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-flexible-plans
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-flexible-plans
$ pip install django-flexible-plans
```

CHAPTER 3

Usage

To use django-flexible-plans in a project, add it to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    ...
    'flexible_plans.apps.FlexiblePlansConfig',
    ...
)
```

Add django-flexible-plans's URL patterns:

```
from flexible_plans import urls as flexible_plans_urls


urlpatterns = [
    ...
    url(r'^', include(flexible_plans_urls)),
    ...
]
```

CHAPTER 4

Documentation for the Code

4.1 Plan Models

The Plan Model is a concrete, swappable subclass of the abstract BasePlan.

- BasePlan
- Plan

class flexible_plans.models.plans.**BasePlan**(**args*, ***kwargs*)

BasePlan is TimeStamped, which means it holds a reference to dates of creation and modification. BasePlan is SoftDeletable, which means it becomes deactivated and not removed entirely (to keep a reference to subscribed plans which become soft-deleted and thus no more available).

BasePlan also can be visible and available. Plan is displayed on the list of currently available plans for user if it is visible. User cannot change plan to a plan that is not visible. Available means that user can buy a plan. If plan is not visible but still available it means that user which is using this plan already will be able to extend this plan again. If plan is not visible and not available, he will be forced then to change plan next time he extends an account.

BasePlan is defined by its Features, which can be quotas, permissions on objects and features

static get_default_plan(*cls*)

Shortcut to retrieve the default plan to be assigned if no plan is selected :param cls: :return: default Plan instance

get_features_dict()

Retrieve a Dict of all the plan's feature to pass through validators :return: {feat.codename: feat.value}

static get_provider_choices()

Scans settings to look for configured payment providers and serve them as choice list :return: [PaymentProvider]

class flexible_plans.models.plans.**Plan**(**args*, ***kwargs*)

Single plan defined in the system. A plan can be customized (for specific users), which means that only those users can purchase this plan and have it selected.

exception DoesNotExist

exception MultipleObjectsReturned

class flexible_plans.models.plans.**PlanFeature**(*id*, *plan*, *feature*)

exception DoesNotExist

exception MultipleObjectsReturned

4.2 Feature Models

There are a few Feature concrete, swappable subsclasses of the abstract BaseFeature to represent different types of features with different logic:

- BaseFeature
- Feature
- MeteredFeature
- CumulativeFeature
- PermissionFeature

class flexible_plans.models.features.**BaseFeature**(**args*, ***kwargs*)
BaseFeature Abstract Model defines the feature behaviour

get_validator()
Get the proper validator from the ValidatorPolicy registry :return: Validator object that matches with the feature codename or None

log_usage()
Logs the feature usage, against the correct validator

class flexible_plans.models.features.**CumulativeFeature**(*id*, *name*, *codename*, *description*, *feature_ptr*, *usage*)

exception DoesNotExist

exception MultipleObjectsReturned

class flexible_plans.models.features.**Feature**(**args*, ***kwargs*)
Feature default implementation Feature is a swappable model to allow client to use their own Feature model. It is recommended to subclass BaseFeature to inherit all the behaviours and base fields. Being Feature a base class of other specific Feature Classes, it support an InheritanceManager to return all kinds of objects on Feature querysets

exception DoesNotExist

exception MultipleObjectsReturned

class flexible_plans.models.features.**MeteredFeature**(*id*, *name*, *codename*, *description*, *feature_ptr*, *units*, *usage*)

exception DoesNotExist

exception MultipleObjectsReturned

4.3 Views

All the CRUD views are defined and available. Plus, specific views for the Plan management are available

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/tobia.ghiraldini/django-flexible-plans/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

5.1.4 Write Documentation

django-flexible-plans could always use more documentation, whether as part of the official django-flexible-plans docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/tobia.ghiraldini/django-flexible-plans/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *django-flexible-plans* for local development.

1. Fork the *django-flexible-plans* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-flexible-plans.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv django-flexible-plans
$ cd django-flexible-plans/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 flexible_plans tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/tobia.ghiraldini/django-flexible-plans/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_flexible_plans
```

CHAPTER 6

Credits

6.1 Development Lead

- Tobia Ghiraldini <tobia.ghiraldini@ninjabit.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.1.0 (2019-01-23)

- First release on PyPI.

Python Module Index

f

Index