django-pesapal Documentation

Release 1.2.0

Billy Odero

Sep 29, 2018

Contents

Contents:

CHAPTER 1

django-pesapal

A django port of pesapal payment gateway

1.1 Documentation

The full documentation is at https://django-pesapal.readthedocs.org.

1.2 Quickstart

Install django-pesapal:

```
pip install django-pesapal
```

Then use it in a project:

```
import django_pesapal
```

1. Add *django_pesapal* to your *INSTALLED_APPS* setting like this:

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

2. Include the *django_pesapal* URLconf in your project urls.py like this:

   ```
   url(r'^payments/', include('django_pesapal.urls')),
   ```

3. You can set your own return url by adding this to *settings.py*:

```
PESAPAL_TRANSACTION_DEFAULT_REDIRECT_URL = 'app_name:url_name'  # this needs to␣
↪be a reversible
```

4. Run *python manage.py migrate* to create the models.
5. Create a method that receives payment details and returns the pesapal iframe url:

```
from django_pesapal.views import PaymentRequestMixin

class PaymentView(PaymentRequestMixin):

    def get_pesapal_payment_iframe(self):

        '''
        Authenticates with pesapal to get the payment iframe src
        '''
        order_info = {
            'first_name': 'Some',
            'last_name': 'User',
            'amount': 100,
            'description': 'Payment for X',
            'reference': 2,  # some object id
            'email': 'user@example.com',
        }

        iframe_src_url = self.get_payment_url(**order_info)
        return iframe_src_url
```

6. Once payment has been processed, you will be redirected to an intermediate screen where the user can finish ordering. Clicking the "Finish Ordering" button will check the payment status to ensure that the payment was successful and then redirects the user to *PESAPAL_TRANSACTION_DEFAULT_REDIRECT_URL*.

1.3 Configuration

Setting	Default Value
PESAPAL_DEMO	True
PESAPAL_CONSUMER_KEY	''
PESAPAL_CONSUMER_SECRET	''
PESAPAL_IFRAME_LINK (if PESAPAL_DEMO=True)	'http://demo.pesapal.com/api/PostPesapalDirectOrderV4'
PESAPAL_IFRAME_LINK (if PESAPAL_DEMO=False)	'https://www.pesapal.com/api/PostPesapalDirectOrderV4'
PESAPAL_QUERY_STATUS_LINK (Demo Mode=True)	'http://demo.pesapal.com/API/QueryPaymentDetails'
PESAPAL_QUERY_STATUS_LINK (Demo Mode=False)	'https://www.pesapal.com/API/QueryPaymentDetails'
PESAPAL_OAUTH_CALLBACK_URL	'transaction_completed'
PESAPAL_OAUTH_SIGNATURE_METHOD	'SignatureMethod_HMAC_SHA1'
PESAPAL_TRANSACTION_DEFAULT_REDIRECT_URL	'/'
PESAPAL_TRANSACTION_FAILED_REDIRECT_URL	''
PESAPAL_REDIRECT_WITH_REFERENCE	True
PESAPAL_TRANSACTION_MODEL	'django_pesapal.Transaction'

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-pesapal
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-pesapal
$ pip install django-pesapal
```

CHAPTER 3

Usage

Install django-pesapal:

```
pip install django-pesapal
```

Then use it in a project:

```
import django_pesapal
```

1. Add *django_pesapal* to your *INSTALLED_APPS* setting like this:

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

2. Include the *django_pesapal* URLconf in your project urls.py like this:

```
url(r'^payments/', include('django_pesapal.urls')),
```

3. You can set your own return url by adding this to *settings.py*:

```
PESAPAL_TRANSACTION_DEFAULT_REDIRECT_URL = 'app_name:url_name'  # this needs to␣
↪be a reversible
```

4. Run *python manage.py migrate* to create the models.

5. Create a method that receives payment details and returns the pesapal iframe url:

```
from django_pesapal.views import PaymentRequestMixin

class PaymentView(PaymentRequestMixin):

    def get_pesapal_payment_iframe(self):
```

(continues on next page)

(continued from previous page)

```
        '''
        Authenticates with pesapal to get the payment iframe src
        '''
        order_info = {
            'first_name': 'Some',
            'last_name': 'User',
            'amount': 100,
            'description': 'Payment for X',
            'reference': 2,  # some object id
            'email': 'user@example.com',
        }

        iframe_src_url = self.get_payment_url(**order_info)
        return iframe_src_url
```

6. Once payment has been processed, you will be redirected to an intermediate screen where the user can finish ordering. Clicking the "Finish Ordering" button will check the payment status to ensure that the payment was successful and then redirects the user to *PESAPAL_TRANSACTION_DEFAULT_REDIRECT_URL*.

NOTE: You can override the intermediate (*post_payment.html*) processing template if you need to have a customized look.

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/odero/django-pesapal/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/odero/django-pesapal/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 :)

4.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/django-pesapal.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-pesapal
$ cd django-pesapal/
$ 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 django_pesapal 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.

4.3 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 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/odero/django-pesapal/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

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

CHAPTER 5

Credits

5.1 Development Lead

- Billy Odero

5.2 Contributors

- Trevor Mutune

CHAPTER 6

History

6.1 1.2.0 (2016-12-11)

- Dropped support for Django 1.7
- Fixes and Upgrades to support Django 1.8 - 1.10
- Use Django's UUIDField

6.2 1.1.0 (2016-05-03)

- Support Django 1.9
- Update payment_method field length from 16 to 24
- Remove support for Py33. Support Py35

6.3 1.0.1 (2015-11-21)

- Fix querydict bug

6.4 1.0.0 (2015-11-11)

- Support Django 1.8
- Support Py33 and Py34
- Return proper IPN response

6.5 0.3.4 (2015-08-12)

- Restructure flow to better support IPN processing

6.6 0.3.3 (2015-06-29)

- Setup build had not packaged the templates

6.7 0.3.2 (2015-06-13)

- Fix documentation formatting issues

6.8 0.3.1 (2015-06-13)

- Allow specifying own transaction model
- Pass all transaction info when redirecting
- Update intermediate template

6.9 0.3 (2015-06-12)

- Introduce intermediate payment processing screen
- Update Django version to 1.7+
- Add support to receive and process IPN
- Save all details about the transaction and status

6.10 0.2.1 (2015-04-03)

- Added test sandbox
- Updated Django version
- Updated django-uuidfield

6.11 0.2 (2015-03-17)

- Support anonymous checkouts
- Add support for getting payment status
- Major structural refactoring. Use mixins
- Use Mixins and XML Builder

6.12 0.1.5 (2014-09-25)

- Pin dependencies to specific versions
- Update how imports should be done
- Remove imports from __init__.py

6.13 0.1.4 (2014-09-23)

- Fix import bug. Tests for projects using this fail in Shippable
- Set max Django version to 1.7

6.14 0.1.3 (2014-07-18)

- Packaging for PyPi

6.15 0.1.2 (2014-06-30)

- Fix import bug in urls.py
- Fix how callback url is constructed
- Fix: Live URL uses https

6.16 0.1.1 (2014-06-30)

- Refactor handling of redirect urls. Model validation of transaction and merchant reference
- Rename settings to conf. Set default oauth redirect url
- Add django-uuidfield to dependencies

6.17 0.1.0 (2014-06-30)

- First release on PyPI.