django-payline Documentation

Release dev

Mathieu Agopian

Sep 27, 2017

Contents

Django-payline helps you make payments with Payline quickly and easily.

CHAPTER 1

Design

The way this is done is by creating a *Payline wallet* with the payment information provided by the user, storing this *wallet ID* in a `Wallet` model, and allowing payments to be done using this *wallet*.

Each payment's *transaction ID* is stored in a `Transaction` model.

CHAPTER 2

Code

The source code is available on Github under the 3-clause BSD license.

CHAPTER 3

Installation

Django-payline makes use of class-based views. It's been written for Django 1.3 but compatibility with older versions is provided using the *django-cbv* package.

If you have Django >= 1.3:

```
pip install django-payline
```

If you have Django < 1.3:

```
pip install django-payline django-cbv
```

Then add `payline` to your `INSTALLED_APPS`, and create the necessary tables:

```
python manage.py syncdb
```

CHAPTER 4

Payline API

By default, Payline's "homologation" WSDL will be used for all the API calls. For those to succeed, make sure you have the necessary settings:

- PAYLINE_MERCHANT_ID
- PAYLINE_KEY
- PAYLINE_VADNBR

The first one will be provided to you by a Payline sales person, and the following two are generated from Payline's web admin interface.

To use Payline in production, you need to provide the production merchant ID, API key and VAD contract number (from Payline's production web admin interface), but you also need to point the settings at the production WSDL file.

To do so, you may use the following setting to point at the production WSDL packaged with the app (which isn't the most up to date, but the one tested):

```
from os import path

import payline

wsdl = path.join(path.dirname(payline.__file__), 'DirectPaymentAPI_prod.wsdl')
PAYLINE_WSDL = 'file://%s' % wsdl
```

CHAPTER 5

Usage

You need to add to your project:

- the URLs
- if you need something different than the default scenario, an implementation of the payment process.

Note: Some very basic templates are provided if you need to use or extend them.

First, create an app. Let's call it `payment`:

```
python manage.py startapp payment
```

Add some URLs in `payment/urls.py`:

```
from django.conf.urls.defaults import patterns, url

from payline.views import ViewWallet, CreateWallet, UpdateWallet


urlpatterns = patterns(
    '',
    url(r'^wallet/$', ViewWallet.as_view(), name='view_wallet'),
    url(r'^wallet/new/$', CreateWallet.as_view(), name='create_wallet'),
    url(r'^wallet/update/$', UpdateWallet.as_view(), name='update_wallet'),
)
```

You can now create wallets, update them, view them, and use them:

- `make_payment`: takes an amount in Euros (€), and asks Payline to make a payment from this *wallet*
- `is_valid`: returns True if the card expiry date is in the future
- `expires_this_month`: returns True if the card expires this month
- `transaction_set`: manager that accesses the *transactions* made on this *wallet*

CHAPTER 6

Extension points

`payline.views.CreateWallet` is a CreateView, and `payline.views.UpdateWallet` is an UpdateView. The default wallet form asks for:

- A first and last name
- The card number
- The card type
- The card expiry
- The card cvx code

The default form checks that the expiry date is in the future, obfuscates the card number (before storing it in the database), and makes sure the information are correct (by creating a *Wallet* on the Payline service, using its API) before creating and storing a *Wallet* locally.

This default form is used both for creating and updating the *Wallet*.

If you want to perform extra validation, or modify the logic, just subclass the form, and pass it to the class-based view, as you would normally do.

CHAPTER 7

Advanced usage

Most of the time, there is a *Wallet* linked to the logged in user. Thus, creating, updating or viewing of **this** *Wallet* only should be allowed.

This can easily be done, for example using a mixin, if there's a `wallet` foreign key added to the user's profile, pointing to `payline.models.Wallet`:

```
from payline import views


class GetWalletMixin(object):
    def dispatch(self, request, *args, **kwargs):
        """View current wallet if it exists, or redirect to create view."""
        profile = request.user.get_profile()
        if profile.wallet is None:
            return redirect('create_wallet')
        kwargs['pk'] = profile.wallet.pk
        return super(GetWalletMixin, self).dispatch(request, *args, **kwargs)


class ViewWallet(GetWalletMixin, views.ViewWallet):
    pass
view_wallet = ViewWallet.as_view()


class UpdateWallet(GetWalletMixin, views.UpdateWallet):
    pass
update_wallet = UpdateWallet.as_view()


class CreateWallet(views.CreateWallet):

    def dispatch(self, request, *args, **kwargs):
        """Redirect to update view if wallet exists."""
        profile = request.user.get_profile()
        if profile.wallet is None:
```

```
            return redirect('update_wallet')
        return super(CreateWallet, self).dispatch(request, *args, **kwargs)
create_wallet = CreateWallet.as_view()
```

CHAPTER 8

Changes

- 0.11: translation
- 0.10: properly fake/mock payline for non-integration tests
- 0.9: better validation of the payment card (authorize first)
- 0.8: production WSDL packaged
- 0.7: card expiry test correct even for last day of month
- 0.6: french translation
- 0.5: removed useless ordering on ‘pk’
- 0.4: fixing missing wsdl (for good)
- 0.3: fixing wsdl (again)
- 0.2: missing wsdl file in the distribution
- 0.1: initial version

CHAPTER 9

Indices and tables

- genindex
- modindex
- search