django-zebra Documentation

Release 1.0

Steven Skoczen & Lee Trout

Apr 21, 2017

Contents

Zebra is a library that makes using Stripe with Django even easier.

Please report errors and omissions at https://github.com/GoodCloud/django-zebra/issues

django-zebra contains two components:

- `zebra` which is the core library providing forms, webhook handlers, abstract models, mixins, signals, and template tags that cover most of the Stripe functionality for many implementations.
- `marty` which is an example app that integrates django-zebra and provides a test suite.

CHAPTER 1

Installation

Install django-zebra:

```
pip install django-zebra
```

Edit your Django settings:

```
INSTALLED_APPS += ('zebra',)
STRIPE_SECRET = "YOUR-SECRET-API-KEY"
STRIPE_PUBLISHABLE = "YOUR-PUBLISHABLE-API-KEY"
```

Enable optional settings

See *Configuration* for settings available.

Enable Webhook URLs

This is optional and only if you wish to use zebra's webhook signals.

```
urlpatterns += patterns('',
  url(r'zebra/', include('zebra.urls', namespace="zebra", app_name='zebra')),
)
```

CHAPTER 2

Quick Start

By default, adding zebra to your installed apps will not provide any functionality beyond the webhooks *if* you enable the url pattern.

The library was designed to be as flexible as possible so it doesn't step on your toes or get in your way.

So following along from the installation above you'll discover that running `syncdb` has no affect. At this point you have 2 options:

- Use the mixins and abstract base classes to extend your own models
- Enable the default application (models & admin)

Customizing your existing application with ABCs and mixins

If you already have an existing model for customers you can still use the default zebra application and simply provide your model class (see the next section).

If you want to forego using the default application you can find all the documentation on this site and in the source code which can be viewed in the GitHub repo.

The easiest way to get started is to extend one of the base classes provided.

Check out *zebra.mixins* for a list of all the mixins that come with the zebra library. The *Tutorial* also goes into detail on mixing zebra with an existing application.

Enabling the default application

To enable the default application simply edit your projects settings and include:

```
ZEBRA_ENABLE_APP = True
```

Then run `syncdb` to install the models.

You should now find the zebra application listed in the admin with three default models:

- Customers
- Plans
- Subscriptions

If you already have a model that you want to use for customers simply add the following setting:

```
ZEBRA_CUSTOMER_MODEL = 'your_app.YourCustomerModel'
```

Then in your models file simply update your model in inherit from `zebra.models.StripeCustomer` abstract base class:

```
from zebra.models import StripeCustomer

class YourCustomerModel(StripeCustomer):
    ...
```

Now `YourCustomerModel` will have 2 new attributes, `stripe` and `stripe_customer` along with a new field, stripe_customer_id. If you've decided to go this route, simply replace `zebra.models.Customer` with `your_app.models.YourCustomerModel` in the examples below.

By default `ZEBRA_AUTO_CREATE_STRIPE_CUSTOMERS` is `True` so creating new customers is a breeze by simply using the mixed-in helper. Simply instantiate your customer class and access the `stripe_customer` attribute.

```
>>> import zebra
>>> c = zebra.models.Customer()
>>> c.stripe_customer_id
>>> c.id
>>> c.stripe_customer
<Customer customer id=cus_A0FULkVQwzwlUz at 0x101b2d7d0> JSON: {
  "account_balance": 0,
  "created": 1329775998,
  "id": "cus_A0FULkVQwzwlUz",
  "livemode": false,
  "object": "customer"
}
>>> c.id
1
>>> c.stripe_customer_id
u'cus_A0FULkVQwzwlUz'
```

As you can see in the example above, with the default customer model there is only 1 field so once the model is in a savable state (all required fields containing valid values) simply calling `c.stripe_customer` created a new customer within Stripe, updated the instance with the newly created customer ID and returned the Stripe API customer instance.

Let's charge our new customer $10.

First we must give our customer a credit card:

```
>>> card = {
... 'number': '4242424242424242',
... 'exp_month': '03',
... 'exp_year': '2013',
... 'cvc': '123',
... 'name': 'John Doe'
```

```
... }
>>> stripe_cust = c.stripe_customer
>>> stripe_cust.card = card
>>> stripe_cust.save()
<Customer customer id=cus_A0FULkVQwzwlUz at 0x101b2d7d0> JSON: {
  "account_balance": 0,
  "active_card": {
    "country": "US",
    "cvc_check": "pass",
    "exp_month": 3,
    "exp_year": 2013,
    "last4": "4242",
    "name": "John Doe",
    "object": "card",
    "type": "Visa"
  },
  "created": 1329775998,
  "id": "cus_A0FULkVQwzwlUz",
  "livemode": false,
  "object": "customer"
}
```

Now we can charge John $10 for that pizza we split:

```
>>> stripe.Charge.create(
... amount=1000,
... currency='usd',
... customer=c.stripe_customer_id,
... description='the money you owed me for the pizza'
... )
<Charge charge id=ch_lCSjHD3hAxXcjO at 0x101e0ff10> JSON: {
  "amount": 1000,
  "card": {
    "country": "US",
    "cvc_check": "pass",
    "exp_month": 3,
    "exp_year": 2013,
    "last4": "4242",
    "name": "John Doe",
    "object": "card",
    "type": "Visa"
  },
  "created": 1329776973,
  "currency": "usd",
  "customer": "cus_A0FULkVQwzwlUz",
  "description": "the money you owed me",
  "disputed": false,
  "fee": 59,
  "id": "ch_lCSjHD3hAxXcjO",
  "livemode": false,
  "object": "charge",
  "paid": true,
  "refunded": false
}
```

CHAPTER 3

Contents:

Tutorial

THIS DOCUMENT IS STILL UNDER DEVELOPMENT

This is an example quickstart that makes many assumptions about your development environment. Please report errors and omissions at https://github.com/GoodCloud/django-zebra/issues

Activate Environment

Create or activate your development environment.

For this example we will be using virtual environment and virtual environment wrapper to create a demo environment.

```
mkvirtualenv --no-site-packages --distribute zebra-demo
```

Install Dependencies

`django-zebra` is a library for integrating Stripe payments into Django applications so you will need to install the following dependencies:

- pycurl *(recommended but not required)*
- django
- stripe
- django-zebra

```
pip install pycurl django stripe django-zebra
```

Configure Django

You'll need to include your Stripe account information in your environment and there are 2 supported ways to do this.

1. Environment Variables
2. Django's settings file

Environment Variables

In BASH just export the variables:

```
export STRIPE_PUBLISHABLE=YOUR_PUB_KEY
export STRIPE_SECRET=YOUR_SECRET_KEY
```

Django Settings

Append the following lines to your project's settings.py file:

```
STRIPE_PUBLISHABLE = "YOUR_PUB_KEY"
STRIPE_SECRET = "YOUR_SECRET_KEY"
```

Zebra

Configuration

There are several options for Zebra. To override any of these options simply add them to your `settings.py` with the value your desire.

`zebra.config.options.`**`STRIPE_PUBLISHABLE`**

default: `''`

Required to use the Stripe API.

Your Stripe API publishable key.

`zebra.config.options.`**`STRIPE_SECRET`**

default: `''`

Required to use the Stripe API.

Your Stripe API secret key.

`zebra.config.options.`**`ZEBRA_AUDIT_RESULTS`**

default

```
{
    'active': 'active',
    'no_subscription': 'no_subscription',
    'past_due': 'past_due',
    'suspended': 'suspended',
    'trialing': 'trialing',
    'unpaid': 'unpaid',
```

```
    'cancelled': 'cancelled'
}
```

Dictionary in which the keys are possible responses from Stripe when checking the status of a subscription. Values are returned when the key matches the subscription status returned from Stripe.

---

zebra.config.options.**ZEBRA_AUTO_CREATE_STRIPE_CUSTOMERS**

default: `True`

Defaults to `True` but is only applicable if *ZEBRA_ENABLE_APP* is `True`.

Boolean to control whether accessing `stripe_customer` on *ZEBRA_CUSTOMER_MODEL* automatically creates a stripe customer if one doesn't exist for the instance.

---

zebra.config.options.**ZEBRA_CARD_YEARS**

default: `range(_today.year, _today.year+12)`

List of years used to populate *ZEBRA_CARD_YEARS_CHOICES*.

---

zebra.config.options.**ZEBRA_CARD_YEARS_CHOICES**

default: `[(i,i) for i in ZEBRA_CARD_YEARS]`

List of pairs (Django choices format) to be used in the credit card year field in `StripePaymentForm`.

---

zebra.config.options.**ZEBRA_CUSTOMER_MODEL**

default: `None`

If *ZEBRA_ENABLE_APP* is `True` then the default value is `zebra.Customer`

---

zebra.config.options.**ZEBRA_ENABLE_APP**

default: `False`

Boolean that enables the default models and admin that comes with zebra. Not to be confused with `marty`.

---

zebra.config.options.**ZEBRA_MAXIMUM_STRIPE_CUSTOMER_LIST_SIZE**

default: `100`

Number of customers to return from querying Stripe when running the managment command to delete test users.

---

zebra.config.options.**ZEBRA_ACTIVE_STATUSES**

default: `('active', 'past_due', 'trialing')`

Iterable of strings that should be considered active based on the values in *ZEBRA_AUDIT_RESULTS*.

---

zebra.config.options.**ZEBRA_INACTIVE_STATUSES**

default: `('cancelled', 'suspended', 'unpaid', 'no_subscription')`

Iterable of strings that should be considered inactive based on the values in *ZEBRA_AUDIT_RESULTS*.

zebra.mixins

zebra.models

zebra.signals

Provides the following signals:

V1

- zebra_webhook_recurring_payment_failed
- zebra_webhook_invoice_ready
- zebra_webhook_recurring_payment_succeeded
- zebra_webhook_subscription_trial_ending
- zebra_webhook_subscription_final_payment_attempt_failed
- zebra_webhook_subscription_ping_sent

v2

- zebra_webhook_charge_succeeded
- zebra_webhook_charge_failed
- zebra_webhook_charge_refunded
- zebra_webhook_charge_disputed
- zebra_webhook_customer_created
- zebra_webhook_customer_updated
- zebra_webhook_customer_deleted
- zebra_webhook_customer_subscription_created
- zebra_webhook_customer_subscription_updated
- zebra_webhook_customer_subscription_deleted
- zebra_webhook_customer_subscription_trial_will_end
- zebra_webhook_customer_discount_created
- zebra_webhook_customer_discount_updated
- zebra_webhook_customer_discount_deleted
- zebra_webhook_invoice_created
- zebra_webhook_invoice_updated
- zebra_webhook_invoice_payment_succeeded
- zebra_webhook_invoice_payment_failed
- zebra_webhook_invoiceitem_created
- zebra_webhook_invoiceitem_updated
- zebra_webhook_invoiceitem_deleted
- zebra_webhook_plan_created
- zebra_webhook_plan_updated

- zebra_webhook_plan_deleted
- zebra_webhook_coupon_created
- zebra_webhook_coupon_updated
- zebra_webhook_coupon_deleted
- zebra_webhook_transfer_created
- zebra_webhook_transfer_failed
- zebra_webhook_ping

zebra.utils

zebra.views

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

z

Index

S

Z