dj-stripe Documentation

Release 0.9.0.dev

Daniel Greenfeld

Jul 24, 2017

Contents

1 Contents **3**

1.1 Installation . . . 3
1.2 Usage . . . 5
1.3 Settings . . . 9
1.4 Cookbook . . . 14
1.5 Not in the Cookbook? . . . 17
1.6 Migrating to dj-stripe . . . 17
1.7 Contributing . . . 18
1.8 Credits . . . 20
1.9 History . . . 22

2 Constraints **29**

- Subscription management
- Subscription during registration
- Single-unit purchases
- Works with Django ~=1.9.1, 1.8
- Works with Python 3.4, 2.7
- Works with Bootstrap 3
- Built-in migrations
- Dead-Easy installation
- Leverages in the best of the 3rd party Django package ecosystem.
- *djstripe* namespace so you can have more than one payments related app.
- Documented (Making good progress)
- Tested (Making good progress)

CHAPTER 1

Contents

Installation

Get the distribution

At the command line:

```
$ pip install dj-stripe
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv dj-stripe
$ pip install dj-stripe
```

Or, if you want to develop on `djstripe` itself:

```
$ git clone https://github.com/<yourname>/dj-stripe/
$ python setup.py develop
```

Configuration

Add `djstripe` to your `INSTALLED_APPS`:

```
INSTALLED_APPS += (
    "djstripe",
)
```

Add your stripe keys:

```
STRIPE_PUBLIC_KEY = os.environ.get("STRIPE_PUBLIC_KEY", "<your publishable key>")
STRIPE_SECRET_KEY = os.environ.get("STRIPE_SECRET_KEY", "<your secret key>")
```

Add some payment plans:

```
DJSTRIPE_PLANS = {
    "monthly": {
        "stripe_plan_id": "pro-monthly",
        "name": "Web App Pro ($25/month)",
        "description": "The monthly subscription plan to WebApp",
        "price": 2500,  # $25.00
        "currency": "usd",
        "interval": "month"
    },
    "yearly": {
        "stripe_plan_id": "pro-yearly",
        "name": "Web App Pro ($199/year)",
        "description": "The annual subscription plan to WebApp",
        "price": 19900,  # $199.00
        "currency": "usd",
        "interval": "year"
    }
}
```

Note: Stripe Plan creation

Not all properties listed in the plans above are used by Stripe - i.e 'description', which is used to display the plans description within specific templates.

Although any arbitrary property you require can be added to each plan listed in DJ_STRIPE_PLANS, only specific properties are used by Stripe. The full list of required and optional arguments can be found here.

Note: The display order of the plans

If you prefer the plans to appear (in views) in the order given in the *DJSTRIPE_PLANS* setting, use an *OrderedDict* from the *collections* module in the standard library, rather than an ordinary dict.

Add to the urls.py:

```
url(r'^payments/', include('djstripe.urls', namespace="djstripe")),
```

Run the commands:

```
python manage.py migrate

python manage.py djstripe_init_customers

python manage.py djstripe_init_plans
```

If you haven't already, add JQuery and the Bootstrap 3.0.0+ JS and CSS to your base template:

```
<!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="https://netdna.bootstrapcdn.com/bootstrap/3.3.4/css/
↪bootstrap.min.css">

<!-- Optional theme -->
<link rel="stylesheet" href="https://netdna.bootstrapcdn.com/bootstrap/3.3.4/css/
↪bootstrap-theme.min.css">
```

```
<!-- Latest JQuery (IE9+) -->
<script src="//code.jquery.com/jquery-2.1.4.min.js"></script>

<!-- Latest compiled and minified JavaScript -->
<script src="https://netdna.bootstrapcdn.com/bootstrap/3.3.4/js/bootstrap.min.js"></
↪script>
```

Also, if you don't have it already, add a javascript block to your base.html file:

```
{% block javascript %}{% endblock %}
```

Running Tests

Assuming the tests are run against PostgreSQL:

```
createdb djstripe
pip install -r requirements_test.txt
python runtests.py
```

Usage

Nearly every project breaks payment types into two broad categories, and will support either or both:

1. Ongoing Subscriptions (Well supported)
2. Individual Checkouts (Early, undocumented support)

Ongoing Subscriptions

dj-stripe provides three methods to support ongoing subscriptions:

- Middleware approach to constrain entire projects easily.
- Class-Based View mixin to constrain individual views.
- View decoration to constrain Function-based views.

> **Warning:** **anonymous** users always raise a `ImproperlyConfigured` exception.
>
> When **anonymous** users encounter these components they will raise a `django.core.exceptions.ImproperlyConfigured` exception. This is done because dj-stripe is not an authentication system, so it does a hard error to make it easier for you to catch where content may not be behind authentication systems.

Any project can use one or more of these methods to control access.

Constraining Entire Sites

If you want to quickly constrain an entire site, the `djstripe.middleware.SubscriptionPaymentMiddleware` middleware does the following to user requests:

- **authenticated** users are redirected to `djstripe.views.SubscribeFormView` unless they:

  - have a valid subscription –or–
  - are superusers (user.is_superuser==True) –or–
  - are staff members (user.is_staff==True).
- **anonymous** users always raise a `django.core.exceptions.ImproperlyConfigured` exception when they encounter these systems. This is done because dj-stripe is not an authentication system.

Example:

Step 1: Add the middleware:

```
MIDDLEWARE_CLASSES = (
    ...
    'djstripe.middleware.SubscriptionPaymentMiddleware',
    ...
    )
```

Step 2: Specify exempt URLS:

```
# sample only - customize to your own needs!
# djstripe pages are automatically exempt!
DJSTRIPE_SUBSCRIPTION_REQUIRED_EXCEPTION_URLS = (
    'home',
    'about',
    "[spam]",  # Anything in the dj-spam namespace
)
```

Using this example any request on this site that isn't on the homepage, about, spam, or djstripe pages is redirected to `djstripe.views.SubscribeFormView`.

Note: The extensive list of rules for this feature can be found at https://github.com/pydanny/dj-stripe/blob/master/djstripe/middleware.py.

See also:

- *Settings*

Constraining Class-Based Views

If you want to quickly constrain a single Class-Based View, the `djstripe.decorators.subscription_payment_required` decorator does the following to user requests:

- **authenticated** users are redirected to `djstripe.views.SubscribeFormView` unless they:
  - have a valid subscription –or–
  - are superusers (user.is_superuser==True) –or–
  - are staff members (user.is_staff==True).
- **anonymous** users always raise a `django.core.exceptions.ImproperlyConfigured` exception when they encounter these systems. This is done because dj-stripe is not an authentication system.

Example:

```
# import necessary Django stuff
from django.http import HttpResponse
from django.views.generic import View
from django.contrib.auth.decorators import login_required

# import the wonderful decorator
from djstripe.decorators import subscription_payment_required

# import method_decorator which allows us to use function
# decorators on Class-Based View dispatch function.
from django.utils.decorators import method_decorator


class MyConstrainedView(View):

    def get(self, request, *args, **kwargs):
        return HttpReponse("I like cheese")

    @method_decorator(login_required)
    @method_decorator(subscription_payment_required)
    def dispatch(self, *args, **kwargs):
        return super(MyConstrainedView, self).dispatch(*args, **kwargs)
```

If you are unfamiliar with this technique please read the following documentation here.

Constraining Function-Based Views

If you want to quickly constrain a single Function-Based View, the `djstripe.decorators.subscription_payment_required` decorator does the following to user requests:

- **authenticated** users are redirected to `djstripe.views.SubscribeFormView` unless they:
  - have a valid subscription –or–
  - are superusers (user.is_superuser==True) –or–
  - are staff members (user.is_staff==True).
- **anonymous** users always raise a `django.core.exceptions.ImproperlyConfigured` exception when they encounter these systems. This is done because dj-stripe is not an authentication system.

Example:

```
# import necessary Django stuff
from django.contrib.auth.decorators import login_required
from django.http import HttpResponse

# import the wonderful decorator
from djstripe.decorators import subscription_payment_required

@login_required
@subscription_payment_required
def my_constrained_view(request):
    return HttpReponse("I like cheese")
```

Don't do this!

Described is an anti-pattern. View logic belongs in views.py, not urls.py.

```
# DON'T DO THIS!!!
from django.conf.urls import patterns, url
from django.contrib.auth.decorators import login_required
from djstripe.decorators import subscription_payment_required

from contents import views

urlpatterns = patterns("",

    # Class-Based View anti-pattern
    url(
        r"^content/$",

        # Not using decorators as decorators
        # Harder to see what's going on
        login_required(
            subscription_payment_required(
                views.ContentDetailView.as_view()
            )
        ),
        name="content_detail"
    ),
    # Function-Based View anti-pattern
    url(
        r"^content/$",

        # Example with function view
        login_required(
            subscription_payment_required(
                views.content_list_view
            )
        ),
        name="content_detail"
    ),
)
```

Extending Subscriptions

`CurrentSubscription.extend(*delta*)`

Subscriptions can be extended by using the `CurrentSubscription.extend` method, which takes a positive `timedelta` as its only property. This method is useful if you want to offer time-cards, gift-cards, or some other external way of subscribing users or extending subscriptions, while keeping the billing handling within Stripe.

> **Warning:** Subscription extensions are achieved by manipulating the `trial_end` of the subscription instance, which means that Stripe will change the status to `trialing`.

Settings

DJSTRIPE_DEFAULT_PLAN (=None)

Payment plans default.

Possibly deprecated in favor of model based plans.

DJSTRIPE_INVOICE_FROM_EMAIL (="billing@example.com")

Invoice emails come from this address.

DJSTRIPE_PLANS (={})

Payment plans.

Possibly deprecated in favor of model based plans.

Example:

```
DJSTRIPE_PLANS = {
    "monthly": {
        "stripe_plan_id": "pro-monthly",
        "name": "Web App Pro ($24.99/month)",
        "description": "The monthly subscription plan to WebApp",
        "price": 2499,  # $24.99
        "currency": "usd",
        "interval": "month",
        "image": "img/pro-monthly.png"
    },
    "yearly": {
        "stripe_plan_id": "pro-yearly",
        "name": "Web App Pro ($199/year)",
        "description": "The annual subscription plan to WebApp",
        "price": 19900,  # $199.00
        "currency": "usd",
        "interval": "year",
        "image": "img/pro-yearly.png"
    }
}
```

Note: Stripe Plan creation

Not all properties listed in the plans above are used by Stripe - i.e 'description' and 'image', which are used to display the plans description and related image within specific templates.

Although any arbitrary property you require can be added to each plan listed in DJ_STRIPE_PLANS, only specific properties are used by Stripe. The full list of required and optional arguments can be found here.

DJSTRIPE_PLAN_HIERARCHY (={})

Payment plans levels.

Allows you to set levels of access to the plans.

Example:

```
DJSTRIPE_PLANS = {
    "bronze-monthly": {
        ...
    },
    "bronze-yearly": {
        ...
    },
    "silver-monthly": {
        ...
    },
    "silver-yearly": {
        ...
    },
    "gold-monthly": {
        ...
    },
    "gold-yearly": {
        ...
    }
}

DJSTRIPE_PLAN_HIERARCHY = {
    "bronze": {
        "level": 1,
        "plans": [
            "bronze-monthly",
            "bronze-yearly",
        ]
    },
    "silver": {
        "level": 2,
        "plans": [
            "silver-monthly",
            "silver-yearly",
        ]
    },
    "gold": {
        "level": 3,
        "plans": [
            "gold-monthly",
            "gold-yearly",
        ]
    },
}
```

Use:

```
{% <plan_name>|djstripe_plan_level %}
```

Example:

```
{% elif customer.current_subscription.plan == plan.plan %}
    <h4>Your Current Plan</h4>
{% elif customer.current_subscription|djstripe_plan_level < plan.plan|djstripe_plan_
↪level %}
```

```
    <h4>Upgrade</h4>
{% elif customer.current_subscription|djstripe_plan_level > plan.plan|djstripe_plan_
↪level %}
    <h4>Downgrade</h4>
{% endif %}
```

DJSTRIPE_PRORATION_POLICY (=False)

By default, plans are not prorated in dj-stripe. Concretely, this is how this translates:

1. If a customer cancels their plan during a trial, the cancellation is effective right away.
2. If a customer cancels their plan outside of a trial, their subscription remains active until the subscription's period end, and they do not receive a refund.
3. If a customer switches from one plan to another, the new plan becomes effective right away, and the customer is billed for the new plan's amount.

Assigning `True` to `DJSTRIPE_PRORATION_POLICY` reverses the functioning of item 2 (plan cancellation) by making a cancellation effective right away and refunding the unused balance to the customer, and affects the functioning of item 3 (plan change) by prorating the previous customer's plan towards their new plan's amount.

DJSTRIPE_PRORATION_POLICY_FOR_UPGRADES (=False)

By default, the plan change policy described in item 3 above holds also for plan upgrades.

Assigning `True` to `DJSTRIPE_PRORATION_POLICY_FOR_UPGRADES` allows dj-stripe to prorate plans in the specific case of an upgrade. Therefore, if a customer upgrades their plan, their new plan is effective right away, and they get billed for the new plan's amount minus the unused balance from their previous plan.

DJSTRIPE_SEND_INVOICE_RECEIPT_EMAILS (=True)

By default dj-stripe sends emails for each receipt. You can turn this off by setting this value to `False`.

DJSTRIPE_SUBSCRIPTION_REQUIRED_EXCEPTION_URLS (=())

Used by `djstripe.middleware.SubscriptionPaymentMiddleware`

Rules:

- "(app_name)" means everything from this app is exempt
- "[namespace]" means everything with this name is exempt
- "namespace:name" means this namespaced URL is exempt
- "name" means this URL is exempt
- The entire djstripe namespace is exempt
- If settings.DEBUG is True, then django-debug-toolbar is exempt

Example:

```
DJSTRIPE_SUBSCRIPTION_REQUIRED_EXCEPTION_URLS = (
    "(allauth)",  # anything in the django-allauth URLConf
    "[blogs]",  # Anything in the blogs namespace
    "products:detail",  # A ProductDetail view you want shown to non-payers
    "home",  # Site homepage
)
```

Note: Adding app_names to applications.

To make the `(allauth)` work, you may need to define an app_name in the `include()` function in the URLConf. For example:

```
# in urls.py
url(r'^accounts/', include('allauth.urls',  app_name="allauth")),
```

DJSTRIPE_SUBSCRIBER_MODEL (=settings.AUTH_USER_MODEL)

If the AUTH_USER_MODEL doesn't represent the object your application's subscription holder, you may define a subscriber model to use here. It should be a string in the form of 'app.model'.

Rules:

- DJSTRIPE_SUBSCRIBER_MODEL must have an `email` field. If your existing model has no email field, add an email property that defines an email address to use.
- You must also implement `DJSTRIPE_SUBSCRIBER_MODEL_REQUEST_CALLBACK`.

Example Model:

```
class Organization(models.Model):
    name = CharField(max_length=200, unique=True)
    subdomain = CharField(max_length=63, unique=True, verbose_name="Organization␣
↪Subdomain")
    owner = ForeignKey(settings.AUTH_USER_MODEL, related_name="organization_owner",␣
↪verbose_name="Organization Owner")

    @property
    def email(self):
        return self.owner.email
```

DJSTRIPE_SUBSCRIBER_MODEL_MIGRATION_DEPENDENCY (="__first__")

If the model referenced in DJSTRIPE_SUBSCRIBER_MODEL is not created in the `__first__` migration of an app you can specify the migration name to depend on here. For example: "0003_here_the_subscriber_model_was_added"

DJSTRIPE_SUBSCRIBER_MODEL_REQUEST_CALLBACK (=None)

If you choose to use a custom subscriber model, you'll need a way to pull it from `request`. That's where this callback comes in. It must be a callable that takes a request object and returns an instance of DJSTRIPE_SUBSCRIBER_MODEL

Examples:

middleware.py

```
class DynamicOrganizationIDMiddleware(object):
    """ Adds the current organization's ID based on the subdomain."""

    def process_request(self, request):
        subdomain = parse_subdomain(request.get_host())

        try:
            organization = Organization.objects.get(subdomain=subdomain)
        except Organization.DoesNotExist:
            return TemplateResponse(request=request, template='404.html', status=404)
        else:
            organization_id = organization.id

        request.organization_id = organization_id
```

settings.py

```
def organization_request_callback(request):
    """ Gets an organization instance from the id passed through ``request``"""
    return Organization.objects.get(id=request.organization_id)
```

Note: This callback only becomes active when `DJSTRIPE_SUBSCRIBER_MODEL` is set.

DJSTRIPE_TRIAL_PERIOD_FOR_SUBSCRIBER_CALLBACK (=None)

Used by `djstripe.models.Customer` only when creating stripe customers.

This is called to dynamically add a trial period to a subscriber's plan. It must be a callable that takes a subscriber object and returns the number of days the trial period should last.

Examples:

```
def static_trial_period(subscriber):
    """ Adds a static trial period of 7 days to each subscriber's account."""
    return 7


def dynamic_trial_period(subscriber):
    """
    Adds a static trial period of 7 days to each subscriber's plan,
    unless they've accepted our month-long promotion.
    """

    if subscriber.coupons.get(slug="monthlongtrial"):
        return 30
    else:
        return 7
```

Note: This setting was named `DJSTRIPE_TRIAL_PERIOD_FOR_USER_CALLBACK` prior to version 0.4

DJSTRIPE_WEBHOOK_URL (=r"^webhook/$")

This is where you can set *Stripe.com* to send webhook response. You can set this to what you want to prevent unnecessary hijinks from unfriendly people.

As this is embedded in the URLConf, this must be a resolvable regular expression.

DJSTRIPE_CURRENCIES (=(('usd', 'U.S. Dollars',), ('gbp', 'Pounds (GBP)',), ('eur', 'Euros',)))

A Field.choices list of allowed currencies for Plan models.

Cookbook

This is a list of handy recipes that fall outside the domain of normal usage.

Customer User Model has_active_subscription property

Very useful for working inside of templates or other places where you need to check the subscription status repeatedly. The *cached_property* decorator caches the result of *has_active_subscription* for a object instance, optimizing it for reuse.

```
# -*- coding: utf-8 -*-

from django.contrib.auth.models import AbstractUser
from django.db import models
from django.utils.functional import cached_property

from djstripe.utils import subscriber_has_active_subscription


class User(AbstractUser):

    """ Custom fields go here """

    def __str__(self):
        return self.username

    def __unicode__(self):
        return self.username

    @cached_property
    def has_active_subscription(self):
        """Checks if a user has an active subscription."""
        return subscriber_has_active_subscription(self)
```

Usage:

```
<ul class="actions">
<h2>{{ object }}</h2>
<!-- first use of request.user.has_active_subscription -->
{% if request.user.has_active_subscription %}
    <p>
```

```
        <small>
            <a href="{% url 'things:update' %}">edit</a>
        </small>
    </p>
{% endif %}
<p>{{ object.description }}</p>

<!-- second use of request.user.has_active_subscription -->
{% if request.user.has_active_subscription %}
    <li>
        <a href="{% url 'places:create' %}">Add Place</a>
        <a href="{% url 'places:list' %}">View Places</a>
    </li>
{% endif %}
</ul>
```

Adding a custom plan that is outside of stripe

Sometimes you want a custom plan for per-customer billing. Or perhaps you are providing a special free-for-open-source plan. In which case, *djstripe.settings.PLAN_CHOICES* is your friend:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from django.contrib.auth.models import AbstractUser
from django.db import models
from django.utils.translation import ugettext_lazy as _

from djstripe.settings import PLAN_CHOICES
from djstripe.signals import subscription_made

CUSTOM_CHOICES = (
    ("custom", "Custom"),
)

CUSTOMIZED_CHOICES = PLAN_CHOICES + CUSTOM_CHOICES

class User(AbstractUser):

    plan = models.CharField(_("plan"), choices=CUSTOMIZED_CHOICES)

    def __unicode__(self):
        return self.username


@receiver(subscription_made)
def my_callback(sender, **kwargs):
    # Updates the User record any time the subscription is changed.
    user = User.objects.get(customer__stripe_id=kwargs['stripe_response'].customer)

    # Only update users with non-custom choices
    if user.plan in [x[0] for x in PLAN_CHOICES]:
        user.plan = kwargs['plan']
        user.save()
```

Making individual purchases

On the subscriber's customer object, use the charge method to generate a Stripe charge. In this example, we're using the user with ID=1 as the subscriber.

```
from decimal import Decimal

from django.contrib.auth import get_user_model

from djstripe.models import Customer


user = get_user_model().objects.get(id=1)

customer, created = Customer.get_or_create(subscriber=user)

amount = Decimal(10.00)
customer.charge(amount)
```

Source code for the Customer.charge method is at https://github.com/pydanny/dj-stripe/blob/master/djstripe/models.py#L412-L430

REST API

The subscriptions can be accessed through a REST API. Make sure you have Django Rest Framework installed (https://github.com/tomchristie/django-rest-framework).

The REST API endpoints require an authenticated user. GET will provide the current subscription of the user. POST will create a new current subscription. DELETE will cancel the current subscription, based on the settings.

- **/subscription/ (GET)**
  - **input**
    - None
  - output (200)
    - id (int)
    - created (date)
    - modified (date)
    - plan (string)
    - quantity (int)
    - start (date)
    - status (string)
    - cancel_at_period_end (boolean)
    - canceled_at (date)
    - current_period_end (date)
    - current_period_start (date)
    - ended_at (date)
    - trial_end (date)

        * trial_start (date)
        * amount (float)
        * customer (int)

- **/subscription/ (POST)**
    - **input**
        * stripe_token (string)
        * plan (string)
    - **output (201)**
        * stripe_token (string)
        * plan (string)
- **/subscription/ (DELETE)**
    - **input**
        * None
    - **Output (204)**
        * None

Not in the Cookbook?

Cartwheel Web provides commercial support for dj-stripe and other open source packages.

Migrating to dj-stripe

There are a number of other Django powered stripe apps. This document explains how to migrate from them to **dj-stripe**.

django-stripe-payments

Most of the settings can be used as is, but with these exceptions:

PAYMENT_PLANS vs DJSTRIPE_PLANS

dj-stripe allows for plans with decimal numbers. So you can have plans that are $9.99 instead of just $10. The price in a specific plan is therefore in cents rather than whole dollars

```
# settings.py

# django-stripe-payments way
PAYMENT_PLANS = {
    "monthly": {
        "stripe_plan_id": "pro-monthly",
        "name": "Web App Pro ($25/month)",
        "description": "The monthly subscription plan to WebApp",
        "price": 25,  # $25.00
```

```
        "currency": "usd",
        "interval": "month"
    },
}

# dj-stripe way
DJSTRIPE_PLANS = {
    "monthly": {
        "stripe_plan_id": "pro-monthly",
        "name": "Web App Pro ($24.99/month)",
        "description": "The monthly subscription plan to WebApp",
        "price": 2499,  # $24.99
        "currency": "usd",
        "interval": "month"
    },
}
```

Migrating Settings

TODO

Migrating Data

Issues:

1. **dj-stripe** includes South migrations and **django-stripe-payments** has no database migrations.
2. **dj-stripe** replaces the `payments.models.StripeObject.created_at` field with `django-model-utils` fields of `model_utls.models.TimeStampedModel.created` and `model_utls.models.TimeStampedModel.modified`.

This will require some sort of one-time migration script. If you create one for your own project, please submit it or link to a paste/gist of the code.

See also:

- https://github.com/pydanny/dj-stripe/issues/10.

Migrating Templates

Issue: django-stripe-payments uses Bootstrap 2 and django-forms-bootstrap, while **dj-stripe** uses Bootstrap 3 and eschews the use of Django form libraries in favor of hand-crafted forms.

TODO: Write this.

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:

Types of Contributions

Report Bugs

Report bugs at https://github.com/pydanny/dj-stripe/issues.

If you are reporting a bug, please include:

- The version of python and Django you're running
- Detailed steps to reproduce the bug.

Fix Bugs

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

Implement Features

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

Write Documentation

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

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/pydanny/dj-stripe/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 :)

Get Started!

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

1. Fork the *dj-stripe* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/dj-stripe.git
```

3. Assuming the tests are run against PostgreSQL:

```
$ createdb djstripe
```

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

```
$ mkvirtualenv dj-stripe
$ cd dj-stripe/
$ python setup.py develop
```

5. Create a branch for local development:

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

   Now you can make your changes locally.

6. When you're done making changes, check that your changes pass the tests, including testing other Python versions with tox. runtests will output both command line and html coverage statistics and will warn you if your changes caused code coverage to drop. Note that if your system time is not in UTC, some tests will fail. If you want to ignore those tests, the –skip-utc command line option is available on runtests.py.:

```
$ pip install -r requirements_test.txt
$ python runtests.py
$ tox
```

7. 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
```

8. Submit a pull request through the GitHub website.

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.
3. If the pull request makes changes to a model, include Django migrations (Django 1.7+).
4. The pull request should work for Python 2.7 and 3.4. Check https://travis-ci.org/pydanny/dj-stripe/pull_requests and make sure that the tests pass for all supported Python versions.

Credits

Development Lead

- Daniel Greenfeld <pydanny@gmail.com>
- Buddy Lindsley
- Yasmine Charif (@dollydagr)
- Audrey Roy
- Mahdi Yusuf (@myusuf3)
- Alexander Kavanaugh (@kavdev)

Contributors

- Luis Montiel <luismmontielg@gmail.com>
- Kulbir Singh (@kulbir)
- Dustin Farris (@dustinfarris)
- Liwen S (@sunliwen)
- centrove
- Chris Halpert (@cphalpert)
- Thomas Parslow (@almost)
- Leonid Shvechikov (@shvechikov)
- sromero84
- Peter Baumgartner (@ipmb)
- Vikas (@vikasgulati)
- Colton Allen (@cmanallen)
- Filip Wasilewski (@nigma)
- Martin Hill (@martinhill)
- Michael Thornhill <michael.thornhill@gmail.com>
- Tobias Lorenz (@Tyrdall)
- Ben Whalley
- nanvel
- jRobb (@jamesbrobb)
- Areski Belaid (@areski)
- José Padilla (@jpadilla)
- Ben Murden (@benmurden)
- Philippe Luickx (@philippeluickx)
- Chriss Mejía (@chrissmejia)
- Bill Huneke (@wahuneke)
- Matt Shaw (@unformatt)
- Chris Trengove (@ctrengove)
- Caleb Hattingh (@cjrh)
- Nicolas Delaby (@ticosax)
- Michaël Krens (@michi88)
- Yuri Prezument (@yprez)
- Raphael Deem (@r0fls)
- Irfan Ahmad (@erfaan)

History

0.9.0 (2016-??-??)

- Charge receipts now take *DJSTRIPE_SEND_INVOICE_RECEIPT_EMAILS* into account (Thanks @r0fls)
- Clarified/modified installation documentation (Thanks @pydanny)
- Corrected and revised ANONYMOUS_USER_ERROR_MSG (Thanks @pydanny)
- Added fnmatching to *SubscriptionPaymentMiddleware* (Thanks @pydanny)
- *SubscriptionPaymentMiddleware.process_request()* functionality broken up into multiple methods, making local customizations easier (Thanks @pydanny)

0.8.0 (2015-12-30)

- better plan ordering documentation (Thanks @cjrh)
- added a confirmation page when choosing a subscription (Thanks @chrissmejia, @areski)
- setup.py reverse dependency fix (#258/#268) (Thanks @ticosax)
- Dropped official support for Django 1.7 (no code changes were made)
- Python 3.5 support, Django 1.9.1 support
- Migration improvements (Thanks @michi88)
- Fixed "Invoice matching query does not exist" bug (#263) (Thanks @mthornhill)
- Fixed duplicate content in account view (Thanks @areski)

0.7.0 (2015-09-22)

- dj-stripe now responds to the invoice.created event (Thanks @wahuneke)
- dj-stripe now cancels subscriptions and purges customers during sync if they were deleted from the stripe dashboard (Thanks @unformatt)
- dj-stripe now checks for an active stripe subscription in the `update_plan_quantity` call (Thanks @ctrengove)
- Event processing is now handled by "event handlers" - functions outside of models that respond to various event types and subtypes. Documentation on how to tie into the event handler system coming soon. (Thanks @wahuneke)
- Experimental Python 3.5 support
- Support for Django 1.6 and lower is now officially gone.
- Much, much more!

0.6.0 (2015-07-12)

- Support for Django 1.6 and lower is now deprecated.
- Improved test harness now tests coverage and pep8
- SubscribeFormView and ChangePlanView no longer populate self.error with form errors

- InvoiceItems.plan can now be null (as it is with individual charges), resolving #140 (Thanks @awechsler and @MichelleGlauser for help troubleshooting)
- Email templates are now packaged during distribution.
- sync_plans now takes an optional api_key
- 100% test coverage
- Stripe ID is now returned as part of each model's str method (Thanks @areski)
- Customer model now stores card expiration month and year (Thanks @jpadilla)
- Ability to extend subscriptions (Thanks @TigerDX)
- Support for plan heirarchies (Thanks @chrissmejia)
- Rest API endpoints for Subscriptions [contrib] (Thanks @philippeluickx)
- Admin interface search by email funtionality is removed (#221) (Thanks @jpadilla)

0.5.0 (2015-05-25)

- Began deprecation of support for Django 1.6 and lower.
- Added formal support for Django 1.8.
- Removed the StripeSubscriptionSignupForm
- Removed `djstripe.safe_settings`. Settings are now all located in `djstripe.settings`
- `DJSTRIPE_TRIAL_PERIOD_FOR_SUBSCRIBER_CALLBACK` can no longer be a module string
- The sync_subscriber argument has been renamed from subscriber_model to subscriber
- Moved available currencies to the DJSTRIPE_CURRENCIES setting (Thanks @martinhill)
- Allow passing of extra parameters to stripe Charge API (Thanks @mthornhill)
- Support for all available arguments when syncing plans (Thanks @jamesbrobb)
- charge.refund() now returns the refunded charge object (Thanks @mthornhill)
- Charge model now has captured field and a capture method (Thanks @mthornhill)
- Subscription deleted webhook bugfix
- South migrations are now up to date (Thanks @Tyrdall)

0.4.0 (2015-04-05)

- Formal Python 3.3+/Django 1.7 Support (including migrations)
- Removed Python 2.6 from Travis CI build. (Thanks @audreyr)
- Dropped Django 1.4 support. (Thanks @audreyr)
- Deprecated the `djstripe.forms.StripeSubscriptionSignupForm`. Making this form work easily with both *dj-stripe* and *django-allauth* required too much abstraction. It will be removed in the 0.5.0 release.
- Add the ability to add invoice items for a customer (Thanks @kavdev)
- Add the ability to use a custom customer model (Thanks @kavdev)
- Added setting to disable Invoice receipt emails (Thanks Chris Halpert)

- Enable proration when customer upgrades plan, and pass proration policy and cancellation at period end for upgrades in settings. (Thanks Yasmine Charif)
- Removed the redundant context processor. (Thanks @kavdev)
- Fixed create a token call in change_card.html (Thanks @dollydagr)
- Fix `charge.dispute.closed` typo. (Thanks @ipmb)
- Fix contributing docs formatting. (Thanks @audreyr)
- Fix subscription cancelled_at_period_end field sync on plan upgrade (Thanks @nigma)
- Remove "account" bug in Middleware (Thanks @sromero84)
- Fix correct plan selection on subscription in subscribe_form template. (Thanks Yasmine Charif)
- Fix subscription status in account, _subscription_status, and cancel_subscription templates. (Thanks Yasmine Charif)
- Now using `user.get_username()` instead of `user.username`, to support custom User models. (Thanks @shvechikov)
- Update remaining DOM Ids for Bootstrap 3. (Thanks Yasmine Charif)
- Update publish command in setup.py. (Thanks @pydanny)
- Explicitly specify tox's virtual environment names. (Thanks @audreyr)
- Manually call django.setup() to populate apps registry. (Thanks @audreyr)

0.3.5 (2014-05-01)

- Fixed `djstripe_init_customers` management command so it works with custom user models.

0.3.4 (2014-05-01)

- Clarify documentation for redirects on app_name.
- If settings.DEBUG is True, then django-debug-toolbar is exempt from redirect to subscription form.
- Use collections.OrderedDict to ensure that plans are listed in order of price.
- Add `ordereddict` library to support Python 2.6 users.
- Switch from `__unicode__` to `__str__` methods on models to better support Python 3.
- Add `python_2_unicode_compatible` decorator to Models.
- Check for PY3 so the `unicode(self.user)` in models.Customer doesn't blow up in Python 3.

0.3.3 (2014-04-24)

- Increased the extendability of the views by removing as many hard-coded URLs as possible and replacing them with `success_url` and other attributes/methods.
- Added single unit purchasing to the cookbook

0.3.2 (2014-01-16)

- Made Yasmine Charif a core committer
- Take into account trial days in a subscription plan (Thanks Yasmine Charif)
- Correct invoice period end value (Thanks Yasmine Charif)
- Make plan cancellation and plan change consistently not prorating (Thanks Yasmine Charif)
- Fix circular import when ACCOUNT_SIGNUP_FORM_CLASS is defined (Thanks Dustin Farris)
- Add send e-mail receipt action in charges admin panel (Thanks Buddy Lindsay)
- Add *created* field to all ModelAdmins to help with internal auditing (Thanks Kulbir Singh)

0.3.1 (2013-11-14)

- Cancellation fix (Thanks Yasmine Charif)
- Add setup.cfg for wheel generation (Thanks Charlie Denton)

0.3.0 (2013-11-12)

- Fully tested against Django 1.6, 1.5, and 1.4
- Fix boolean default issue in models (from now on they are all default to *False*).
- Replace duplicated code with *djstripe.utils.user_has_active_subscription*.

0.2.9 (2013-09-06)

- Cancellation added to views.
- Support for kwargs on charge and invoice fetching.
- def charge() now supports send_receipt flag, default to True.
- Fixed templates to work with Bootstrap 3.0.0 column design.

0.2.8 (2013-09-02)

- Improved usage documentation.
- Corrected order of fields in StripeSubscriptionSignupForm.
- Corrected transaction history template layout.
- Updated models to take into account when settings.USE_TZ is disabled.

0.2.7 (2013-08-24)

- Add handy rest_framework permission class.
- Fixing attribution for django-stripe-payments.
- Add new status to Invoice model.

0.2.6 (2013-08-20)

- Changed name of division tag to djdiv.
- Added `safe_setting.py` module to handle edge cases when working with custom user models.
- Added cookbook page in the documentation.

0.2.5 (2013-08-18)

- Fixed bug in initial checkout
- You can't purchase the same plan that you currently have.

0.2.4 (2013-08-18)

- Recursive package finding.

0.2.3 (2013-08-16)

- Fix packaging so all submodules are loaded

0.2.2 (2013-08-15)

- Added Registration + Subscription form

0.2.1 (2013-08-12)

- Fixed a bug on CurrentSubscription tests
- Improved usage documentation
- Added to migration from other tools documentation

0.2.0 (2013-08-12)

- Cancellation of plans now works.
- Upgrades and downgrades of plans now work.
- Changing of cards now works.
- Added breadcrumbs to improve navigation.
- Improved installation instructions.
- Consolidation of test instructions.
- Minor improvement to django-stripe-payments documentation
- Added coverage.py to test process.
- Added south migrations.
- Fixed the subscription_payment_required function-based view decorator.

- Removed unnecessary django-crispy-forms

0.1.7 (2013-08-08)

- Middleware excepts all of the djstripe namespaced URLs. This way people can pay.

0.1.6 (2013-08-08)

- Fixed a couple template paths
- Fixed the manifest so we include html, images.

0.1.5 (2013-08-08)

- Fixed the manifest so we include html, css, js, images.

0.1.4 (2013-08-08)

- Change PaymentRequiredMixin to SubscriptionPaymentRequiredMixin
- Add subscription_payment_required function-based view decorator
- Added SubscriptionPaymentRedirectMiddleware
- Much nicer accounts view display
- Much improved subscription form display
- Payment plans can have decimals
- Payment plans can have custom images

0.1.3 (2013-08-7)

- Added account view
- Added Customer.get_or_create method
- Added djstripe_sync_customers management command
- sync file for all code that keeps things in sync with stripe
- Use client-side JavaScript to get history data asynchronously
- More user friendly action views

0.1.2 (2013-08-6)

- Admin working
- Better publish statement
- Fix dependencies

0.1.1 (2013-08-6)

- Ported internals from django-stripe-payments
- Began writing the views
- Travis-CI
- All tests passing on Python 2.7 and 3.3
- All tests passing on Django 1.4 and 1.5
- Began model cleanup
- Better form
- Provide better response from management commands

0.1.0 (2013-08-5)

- First release on PyPI.

CHAPTER 2

Constraints

1. For stripe.com only
2. Only use or support well-maintained third-party libraries
3. For modern Python and Django