dinero Documentation

Release 0.0.1

Fusionbox

April 06, 2016

Contents

Dinero is library for Python that provides a clean minimal interface for payment processing. It abstracts the differences between different payment gateways in order for you not to have to learn each one in depth.

CHAPTER 1

Usage

1.1 Payments 101

Note: This document aims to introduce a developer to what is involved in taking online payments. If you already have experience with online payments, then you can probably skip ahead to the Quickstart.

There are a lot of pieces involved in accepting payments. The first thing you need in order to accept payments is a merchant account. Then you need a payment gateway. Then you need code which talks to the payment gateway. Associated with the merchant account is a payment processor. The payment processor talks to credit card companies. All of these pieces add up together in order for an end user to enter their credit card and you to receive money from them.

There seem to be two paradigms in online payments. One is the PayPal paradigm, where customers are redirected off site to the PayPal site where they log into their PayPal account and confirm the payment on that site and then are redirected back to your website afterwards. Examples of PayPal-style payment providers include, of course, PayPal, Amazon Payments, and Google Wallet.

The other paradigm is sometimes called the server-to-server paradigm. With server-to-server, users are not redirected off of your site, your server code will communicate the necessary data to the gateway behind the scenes. Server-to-server payments give you (the developer) complete control of the customer experience. Server-to-server payments also give you more control of information management and recurring payments. Examples of server-to-server providers include Authorize.Net, Braintree payments, and Stripe.

1.1.1 Merchant Account

A merchant account is a special type of bank account that can receive credit card transactions. To protect against fraud, opening a merchant account is something of an involved process.

Todo

Explain how to get a merchant account???

1.1.2 Payment Gateway

Payment gateways are what communicate between the credit card processors and your merchant account. There are several credit card processors these days, but the major ones include Authorize.Net, Braintree Payments, and Stripe.

Braintree and Stripe both offer accounts that don't require merchant accounts. If you are familiar with Square, this is quite similar. These accounts are more convenient in that you don't have to go through the hassle of getting a merchant account and you don't have to pay the merchant account fees. However, the pricing for this type of account is generally higher per transaction (almost always 2.9% + 30¢ per transaction).

Stripe and merchant account-less Braintree accounts are good for companies that are just starting out who don't expect to have large transactions. Often it is cheaper to have one of these accounts because the high transaction fees are still smaller than the monthly fees associated with merchant accounts.

Authorize.Net or merchant account-backed Braintree accounts are more suitable for website that expect high volume or high priced transactions, or for companies that also want to do offline payments as well.

Features

Payment gateways usually offer a variation on the same list of features.

Transactions

The usual process for transaction is as follows:

1. Send a payment to the gateway. The gateway will validate the payment and then send it to the processor who will validate that the credit card is real and "authorize" the payment. Validating means checking that the credit card number is real, that the CVV code matches or that the address is correct. Authorization is a fancy word for making sure that the credit card actually has money for the payment.
2. Then you submit the payment for settlement. This means that you wish the payment to actually go through. Payments are usually settled once a day. Settlement is the actually process of transferring the money.

Prior to a payment being settled, it can be voided. If a payment is neither settled nor voided within 30 days, it goes away. A processor may charge you a fee if you leave a payment suspended like that.

After a payment has been settled, you may refund all or part of a payment. This process takes the money from your merchant account and gives it back to the customer.

Note: It is possible to charge a credit card with only the number and the expiration date. Name on card, CVV code, and billing address are all optional fields. However, if you ask for CVV code and/or billing address you can verify your transaction more soundly.

With certain payment gateways, more verified qualify for better rates. There is a lot of risk involved for the payment companies and they will charge more when they are worried about fraud.

Vaulting

If you store a customer's credit card information on your server, you are exposing yourself to some big liabilities. There is this thing called PCI compliance which is sort of a list of regulations that you need to conform to when processing credit cards. It is much preferable to store that information with the gateway, who can afford those risks. The vaulting process is something similar to the following:

1. Collect the customer's credit card information on your website. It is especially important to avoid storing (or logging!) the credit card number or the CVV code.
2. Send the information to the gateway. The gateway will give you a token or and ID that you can use to reference the credit card. You don't have access to the credit card number anymore, but that is probably for the best.

Note: Some payment gateways offer solutions for storing credit card information that never need to touch your server. These are very convenient because they may help avoid the need for you to have a PCI compliant website.

Older implementations of this included redirects to the gateway website to display the form. This may not be acceptable for some websites because it makes it difficult to control the customer experience and also to track the customer.

Newer solutions include JavaScript libraries that allow you to capture the credit card information in the browser and communicate to the gateway over AJAX. This allows you to have complete control over the interface, but may not be the perfect solution for everyone.

Alternatives

There are some alternative online payment providers that have been cropping up recently. Because of the hassles involved with credit card risk and dealing with credit card processors, there are some companies like Dwolla or GoCardless, that are skipping credit cards and connect directly to your bank account for payments. These providers seems to have drastically lower fees, but at the downside of requiring a user to enter their bank account information.

Additionally, if you need to make bi-directional payments, it is kind of difficult with the traditional gateway. Balanced Payments and Stripe Connect are geared more towards marketplaces where the website collects money for its users.

1.2 Quickstart

1.2.1 1. Configure

The first thing you need to do in order to use dinero is to configure your gateways(s). The following example would be configuration for an Authorize.Net gateway:

```
import dinero

dinero.configure({
    # a name that you can remember
    'auth.net': {
        'type': 'dinero.gateways.AuthorizeNet',
        'default': True,
        # Gateway specific configuration
        'login_id': 'XXX',
        'transaction_key': 'XXX',
    }
})
```

1.2.2 2. Make Transactions

Now that you have a gateway configured, you can create transactions.

```
transcation = dinero.Transaction.create(
    price=2000,
    number='4111111111111111',
    month='12',
    year='2012',
)
```

1.2.3 3. Profit

Well that's up to you now isn't it.

1.3 Gateways

A payment gateway is what takes the credit card information and changes that into money in your bank account. Dinero currently supports Authorize.Net and has some support for Braintree Payments.

In order to use dinero, you must first configure a gateway. The basic configuration looks like:

```
import dinero

dinero.configure({
    'foo': {
        'type': 'XXX',
        'default': True,
        # ...
    },
})
```

where `foo` is a reference name for you to remember. The `type` is the class that implements the gateway. Dinero currently has the following gateway types:

- *dinero.gateways.AuthorizeNet*
- `dinero.gateways.Braintree` (incomplete implementation)

The gateway marked `default` will be used by default when creating transactions.

class `dinero.gateways.`**`AuthorizeNet`**

The Authorize.Net gateway requires the following packages.

- requests
- lxml

In order to configure the Authorize.Net gateway, you need the Login ID and the Transaction Key.

```
import dinero

dinero.configure({
    'foo': {
        'type': 'dinero.gateways.AuthorizeNet',
        'default': True,
        'login_id': 'XXX',
        'transaction_key': 'XXX',
    },
})
```

1.4 Transactions

Transaction objects contain data about payments. Every transaction object has a `transaction_id` and a `price`.

You can create a basic credit card transaction by using *Transaction.create()*:

```
>>> import dinero
>>> transaction = dinero.Transaction.create(
        price=200,
        number='4111111111111111',
        month='12',
        year='2015',
    )
>>> transaction.transaction_id
'0123456789'
```

This will charge the credit card $200. If you store the `transaction_id`, you can later retrieve the transaction object.

```
>>> transaction = dinero.Transaction.retrieve('0123456789')
>>> transaction.price
200
```

Note: Like many methods in dinero, *`Transaction.create()`* and *`Transaction.retrieve()`* accept a `gateway_name` parameter. This parameter corresponds with the gateway name that you created when configuring your gateways.

If you had the following configuration:

```
import dinero
dinero.configure({
    'new-auth.net': {
        'type': 'dinero.gateways.AuthorizeNet',
        'default': True,
        ...
    },
    'old-auth.net': {
        'type': 'dinero.gateways.AuthorizeNet',
        ...
    },
})
```

If you don't specify `gateway_name`, it will use `new-auth.net`. If you wanted to use `old-auth.net`, you could do something like the following:

```
dinero.Transaction.create(
    gateway_name='old-auth.net',
    price=200,
    ...
)
```

When you have a transaction object, you can refund it:

```
transaction.refund()
```

If a transaction has not yet been settled, the transaction will simply be voided, otherwise an actual refund will take place. If a transaction has settled, you can pass refund the optional `amount` argument, in case you only want to give a partial refund.

```
transaction.refund(100)
```

1.4.1 Delayed Settlement

By default, dinero will automatically submit a transaction for settlement, however you can override this by setting the `settle` argument to `False`. When you need to settle a transaction, you can call *`Transaction.settle()`*:

```
transaction = dinero.Transaction.create(
    price=200,
    number='4111111111111111',
    month='12',
    year='2015',
    settle=False,
)

...

transaction.settle()
```

If you need to cancel a transaction instead of settling it, just call *`Transaction.refund()`*.

1.4.2 API

class `dinero.`**`Transaction`**(*gateway_name*, *price*, *transaction_id*, ***kwargs*)

`Transaction` is an abstraction over payments in a gateway. This is the interface for creating payments.

classmethod `create`(*price*, ***kwargs*)

Creates a payment. This method will actually charge your customer. *`create()`* can be called in several different ways.

You can call this with the credit card information directly.

```
Transaction.create(
    price=200,
    number='4111111111111111',
    year='2015',
    month='12',

    # optional
    first_name='John',
    last_name='Smith,'
    zip='12345',
    address='123 Elm St',
    city='Denver',
    state='CO',
    cvv='900',
    email='johnsmith@example.com',
)
```

If you have a *`dinero.Customer`* object, you can create a transaction against the customer.

```
customer = Customer.create(
    ...
)

Transaction.create(
    price=200,
    customer=customer,
)
```

Other payment options include `card` and `check`. See *`dinero.CreditCard`* for more information.

classmethod **retrieve**(*transaction_id*[, *gateway_name=None*])
Fetches a transaction object from the gateway.

refund([*amount=None*])
If `amount` is None dinero will refund the full price of the transaction.

Payment gateways often allow you to refund only a certain amount of money from a transaction. Refund abstracts the difference between refunding and voiding a payment so that normally you don't need to worry about it. However, please note that you can only refund the entire amount of a transaction before it is settled.

settle([*amount=None*])
If you create a transaction without settling it, you can settle it with this method. It is possible to settle only part of a transaction. If `amount` is None, the full transaction price is settled.

1.5 Customers

Payment gateways allow you to store information about your customers. They let you store credit cards securely so that you can remember cards without actually storing the sensitive information on your server. If your database is compromised you won't leak all of your customers' information.

We have two objects that you can use to manage your customers' data.

1.5.1 Customers

The *Customer* class provides an interface quite similar to *Transaction*. To create a customer, you use *Customer.create()*:

```
>>> customer = Customer.create(
        email='bill@example.com',
        number='4111111111111111',
        cvv='900',
        address='123 Elm St',
        zip='12345',
    )
>>> customer.customer_id
'1234567890'
>>> customer.card_id
'0000101010'
```

Todo

Are the credit card fields required when creating a Customer? Dinero doesn't really require it, but Authorize.Net seems to require you put either a credit card or a bank account (see page 14 of Authorize.Net's CIM XML Guide).

Similarly, you can also retrieve customers. However, whereas transactions are not really editable, if you want to update a customer's information you can. Just call the *Customer.save()* method when you have made your changes.

```
>>> customer = Customer.retrieve('1234567890')
>>> customer.email = 'fred@example.com'
>>> customer.save()
```

You wouldn't really have a use for storing a customer's payment data if you weren't actually going to use it. If you want to charge a customer, *Transaction.create()* accepts a *Customer* object:

```
customer = Customer.retrieve('1234567890')

transaction = Transaction.create(
    price=200,
    customer=customer,
)
```

1.5.2 Credit Cards

Every *Customer* also has list of credit cards that can be accessed at `customer.cards`.

When you create your Customer, it will create the first card:

```
>>> customer = Customer.create(
        email='bill@example.com',
        number='4111111111111111',
        cvv='900',
        address='123 Elm St',
        zip='12345',
    )
>>> card = customer.cards[0]
>>> card.last_4
'1111'
```

If you have a secondary card, you can add it using *Customer.add_card()*.

```
customer.add_card(
    first_name='John',
    last_name='Smith',
    number='4222222222222',
    cvv='900',
    address='123 Elm St',
    zip='12345',
)
```

The *CreditCard* class is editable like *Customer*:

```
card.first_name = 'Fred'
card.save()
```

Note: When you create a *CreditCard*, it will be validated. This is quite useful if you are going to store a credit card and charge it later when you don't have access to the user to fix the information.

`address` and `zip` are required by Visa when doing a Zero-Dollar Authorization. This is a special process for validating that a card is real without actually charging money to it. For other credit card types, 1¢ is usually charged and immediately voided when validating a credit card.

When you are testing your payments application, you may need to input credit cards that validate. Here is a list of test credit card numbers.

1.5.3 API

Todo

Provide a list of fields that an instance will always have for Customer and CreditCard.

Customer

- customer_id
- first_name
- last_name
- email
- cards

Card

- customer_id
- card_id
- last_4

---

class dinero.**Customer**(*customer_id*, ***kwargs*)

A *Customer* object stores information about your customers.

classmethod create(*email*, ***kwargs*)

Creates and stores a customer object. When you first create a customer, you are required to also pass in arguments for a credit card.

```
Customer.create(
    email='bill@example.com',

    # required for credit card
    number='4111111111111111',
    cvv='900',
    month='12',
    year='2015',
    address='123 Elm St.',
    zip='12345',
)
```

This method also accepts `gateway_name`.

classmethod retrieve(*customer_id*[, *gateway_name=None*])

Fetches a customer object from the gateway. This optionally accepts a `gateway_name` parameter.

save()

Saves changes to a customer object.

delete()

Deletes a customer object from the gateway.

cards

Contains a list of all the cards associated with a customer. This is populated by *create()* and *retrieve()* and appended to by *add_card()*.

add_card(**args*, ***kwargs*)

The first credit card is added when you call *create()*, but you can add more cards using this method.

```
customer.add_card(
    number='4222222222222',
    cvv='900',
    month='12'
    year='2015'
    address='123 Elm St',
```

```
            zip='12345',
        )
```

class dinero.**CreditCard**(*customer_id*, *card_id*, ***kwargs*)

A representation of a credit card to be stored in the gateway.

save()

Save changes to a card to the gateway.

delete()

Delete a card from the gateway.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

A

C

D

R

S

T