django-getpaid Documentation

Release 2.2.2

Sunscrapers

Jun 03, 2021

CONTENTS

1 Contents: 3

2 Development team 19

3 Indices and tables 21

Index 23

django-getpaid is a multi-broker payment processor for Django. Main features include:

- support for multiple payment brokers at the same time
- very flexible architecture
- support for asynchronous status updates - both push and pull
- support for modern REST-based broker APIs
- support for multiple currencies (but one per payment)
- support for global and per-plugin validators
- easy customization with provided base abstract models and swappable mechanic (same as with Django's User model)

We would like to provide a *catalog* of plugins for `django-getpaid` - if you create a plugin please let us know.

Disclaimer: this project has nothing in common with getpaid plone project.

This project uses semantic versioning.

CHAPTER

ONE

CONTENTS:

1.1 Installation & Configuration

This document presents the minimal steps required to use `django-getpaid` in your project.

1.1.1 Get it from PyPI

```
pip install django-getpaid
```

We do not recommend using development version as it may contain bugs.

1.1.2 Install at least one plugin

There should be several plugins available in our repo. Each follows this schema: `django-getpaid-<backend_name>` For example if you want to install PayU integration, run:

```
pip install django-getpaid-payu
```

1.1.3 Enable app and plugin

Next, add `"getpaid"` and any plugin to `INSTALLED_APPS` in your `settings.py`. Plugins have the format `getpaid_<backend_name>`:

```
INSTALLED_APPS = [
    # ...
    "getpaid",
    "getpaid_payu",
]
```

1.1.4 Create Order model

You need to create your own model for an order. It should inherit from `getpaid.models.AbstractOrder` (if not, it must implement its methods) and you need to implement some methods. It could look like this example:

```
from django.conf import settings
from django.db import models
from getpaid.models import AbstractOrder

class CustomOrder(AbstractOrder):
    buyer = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    description = models.CharField(max_length=128, default='Order from mystore')
    total = models.DecimalField()
    currency = models.CharField(max_length=3, default=settings.DEFAULT_CURRENCY)

    def get_buyer_info(self):
        return {"email": self.buyer.email}

    def get_total_amount(self):
        return self.total

    def get_description(self):
        return self.description

    def get_currency(self):
        return self.currency

    # either one of those two is required:
    def get_redirect_url(self, *args, success=None, **kwargs):
        # this method will be called to get the url that will be displayed
        # after returning from the paywall page and you can use `success` param
        # to differentiate the behavior in case the backend supports it.
        # By default it returns this:
        return self.get_absolute_url()

    def get_absolute_url(self):
        # This is a standard method recommended in Django documentation.
        # It should return an URL with order details. Here's an example:
        return reverse("order-detail", kwargs={"pk": self.pk})

    # these are optional:
    def is_ready_for_payment(self):
        # Most of the validation will be handled by the form
        # but if you need any extra logic beyond that, you can write it here.
        # This is the default implementation:
        return True

    def get_items(self):
        # Some backends expect you to provide the list of items.
        # This is the default implementation:
        return [{
            "name": self.get_description(),
            "quantity": 1,
            "unit_price": self.get_total_amount(),
        }]
```

1.1.5 Tell `getpaid` what model handles orders

Put this inside your `settings.py`:

```
GETPAID_ORDER_MODEL = "yourapp.CustomOrder"
```

1.1.6 (Optional) Provide custom Payment model

If you want, you can provide your own Payment model. Read more in *Customization - Payment API*.

Note: Payment model behaves like django.auth.User model - after you use the original, migration to a custom version is VERY hard.

1.1.7 Add getpaid to urls

```
urlpatterns = [
    # ...
    path("payments", include("getpaid.urls")),
]
```

1.1.8 Provide config for plugins

For each installed plugin you can configure it in `settings.py`:

```
GETPAID = {
    "BACKENDS":{
        "getpaid_payu": {   # dotted import path of the plugin
            # refer to backend docs and take these from your merchant panel:
            "pos_id": 12345,
            "second_key": "91ae651578c5b5aa93f2d38a9be8ce11",
            "client_id": 12345,
            "client_secret": "12f071174cb7eb79d4aac5bc2f07563f",
        },

        # this plugin is meant only for testing purposes
        "getpaid.backends.dummy": {
            "confirmation_method": "push",
        },
    }
}
```

1.1.9 Prepare views and business logic

The logic for building an order is up to you. You can eg. use a cart application to gather all Items for your Order.

An example view and its hookup to urls.py can look like this:

```
# orders/views.py
from getpaid.forms import PaymentMethodForm

class OrderView(DetailView):
    model = Order

    def get_context_data(self, **kwargs):
        context = super(OrderView, self).get_context_data(**kwargs)
        context["payment_form"] = PaymentMethodForm(
            initial={"order": self.object, "currency": self.object.currency}
        )
        return context

# main urls.py

urlpatterns = [
    # ...
    path("order/<int:pk>/", OrderView.as_view(), name="order_detail"),
]
```

You'll also need a template (`order_detail.html` in this case) for this view. Here's the important part:

```
<h2>Choose payment broker:</h2>
<form action="{% url 'getpaid:create-payment' %}" method="post">
  {% csrf_token %}
  {{ payment_form.as_p }}
  <input type="submit" value="Checkout">
</form>
```

And that's pretty much it.

After you open order detail you should see a list of plugins supporting your currency and a "Checkout" button that will redirect you to selected paywall. After completing the payment, you will return to the same view.

Please see fully working example app.

1.1.10 Next steps

If you're not satisfied with provided Payment model or the PaymentMethodForm, please see *customization docs*.

1.2 Plugin catalog

All plugins in alphabetical order.

Currently a lot of plugins is being developed or ported. Please stay tuned.

1.2.1 PayU

- PyPI name: `django-getpaid-payu`
- Repository: https://github.com/django-getpaid/django-getpaid-payu
- Original API Docs: https://www.payu.pl/en/developers
- Sandbox panel: https://merch-prod.snd.payu.com/user/login
- Currencies: BGN, CHF, CZK, DKK, EUR, GBP, HRK, HUF, NOK, PLN, RON, RUB, SEK, UAH, USD

1.3 Settings

- *Core settings*
- *Backend settings*
- *Optional settings*

1.3.1 Core settings

GETPAID_ORDER_MODEL

No default, you **must** provide this setting.

The model to represent an Order. See *Customization - Payment API*.

> **Warning:** You cannot change the `GETPAID_ORDER_MODEL` setting during the lifetime of a project (i.e. once you have made and migrated models that depend on it) without serious effort. It is intended to be set at the project start, and the model it refers to must be available in the first migration of the app that it lives in.

GETPAID_PAYMENT_MODEL

Default: `'getpaid.Payment'`

The model to represent a Payment. See *Customization - Payment API*.

> **Warning:** You cannot change the `GETPAID_PAYMENT_MODEL` setting during the lifetime of a project (i.e. once you have made and migrated models that depend on it) without serious effort. It is intended to be set at the project start, and the model it refers to must be available in the first migration of the app that it lives in.

1.3.2 Backend settings

To provide configuration for payment backends, place them inside `GETPAID_BACKEND_SETTINGS` dictionary. Use plugin's dotted path - just as you put it in `INSTALLED_APPS` - as a key for the config dict. See this example:

```
GETPAID_BACKEND_SETTINGS = {
    "getpaid.backends.dummy": {
        "confirmation_method": "push",
        "gateway": reverse_lazy("paywall:gateway"),
    },
    "getpaid_paynow": {
        "api_key": "9bcdead5-b194-4eb5-a1d5-c1654572e624",
        "signature_key": "54d22fdb-2a8b-4711-a2e9-0e69a2a91189",
    },
}
```

Each backend defines its own settings this way. Please check the backend's documentation.

1.3.3 Optional settings

A place for optional settings is `GETPAID` dictionary, empty by default. It can contain these keys:

POST_TEMPLATE

Default: None

This setting is used by processor's default `get_template_names()` method to override backend's `template_name`. The template is used to render that backend's POST form. This setting can be used to provide a global default for such cases if you use more plugins requiring such template. You can also use `POST_TEMPLATE` key in *backend's config* to override the template just for one backend.

POST_FORM_CLASS

Default: None

This setting is used by backends that use POST flow. This setting can be used to provide a global default for such cases if you use more plugins requiring such template. You can also use `POST_FORM_CLASS` key in *backend's config* to override the template just for one backend. Use full dotted path name.

SUCCESS_URL

Default: `"getpaid:payment-success"`

Allows setting custom view name for successful returns from paywall. Again, this can also be set on a per-backend basis.

If the view requires kwargs to be resolved, you need to override

FAILURE_URL

Default: `"getpaid:payment-failure"`

Allows setting custom view name for fail returns from paywall. Again, this can also be set on a per-backend basis.

HIDE_LONELY_PLUGIN

Default: False

Allows you to hide plugin selection if only one plugin would be presented. The hidden plugin will be chosen as default.

VALIDATORS

Default: []

Here you can provide import paths for validators that will be run against the payment before it is sent to the paywall. This can also be set on a per-backend basis.

1.4 Customization - Payment API

Django-getpaid was designed to be very customizable. In this document you'll read about the Payment API which lets you customize most of the mechanics of `django-getpaid`.

Since most Payment methods act as interface to PaymentProcessor, you can use this to add extra layer between the Payment and the PaymentProcessor.

1.4.1 Basic Order API

class getpaid.models.**AbstractOrder**(**args*, ***kwargs*)

Please consider setting either primary or secondary key of your Orders to UUIDField. This way you will hide your volume which is valuable business information that should be kept hidden. If you set it as secondary key, remember to use `dbindex=True` (primary keys are indexed by default). Read more: https://docs.djangoproject.com/en/3.0/ref/models/fields/#uuidfield

get_return_url(**args*, *success=None*, ***kwargs*) → str

Method used to determine the final url the client should see after returning from gateway. Client will be redirected to this url after backend handled the original callback (i.e. updated payment status) and only if SUCCESS_URL or FAILURE_URL settings are NOT set. By default it returns the result of *get_absolute_url*

get_absolute_url() → str

Standard method recommended in Django docs. It should return the URL to see details of particular Payment (or usually - Order).

is_ready_for_payment() → bool

Most of the validation is made in PaymentMethodForm but if you need any extra validation. For example you most probably want to disable making another payment for order that is already paid.

You can raise `ValidationError` if you want more verbose error message.

get_items() → List[getpaid.types.ItemInfo]

There are backends that require some sort of item list to be attached to the payment. But it's up to you if the list is real or contains only one item called "Payment for stuff in {myshop}" ;)

Returns List of `ItemInfo` dicts. Default: order summary.

Return type List[ItemInfo]

get_total_amount() → decimal.Decimal
This method must return the total value of the Order.

Returns Decimal object

get_buyer_info() → getpaid.types.BuyerInfo
This method should return dict with necessary user info. For most backends email should be sufficient. Refer to :class`BuyerInfo` for expected structure.

get_description() → str

Returns Description of the Order. Should return the value of appropriate field.

1.4.2 Basic Payment API

AbstractPayment defines a minimal set of fields that are expected by *BaseProcessor* API. If you want to have it completely your own way, make sure to provide properties linking your fieldnames to expected names.

class getpaid.models.**AbstractPayment**(**args*, ***kwargs*)

id
UUID to not disclose your volume.

order
ForeignKey to (swappable) Order model.

amount_required
Decimal value with 4 decimal places. Total value of the Order that needs to be paid.

currency
Currency code in ISO 4217 format.

status
Status of the Payment - one of `PAYMENT_STATUS_CHOICES`. This field is managed using django-fsm.

backend
Identifier of the backend processor used to handle this Payment.

created_on
Datetime of Payment creation - automated.

last_payment_on
Datetime the Payment has been completed. Defaults to NULL.

amount_paid
Amount actually paid by the buyer. Should be equal amount_required if backend does not support partial payments. Will be smaller than that after partial refund is done.

amount_locked
Amount that has been pre-authed by the buyer. Needs to be charged to finalize payment or released if the transaction cannot be fulfilled.

amount_refunded
Amount that was refunded. Technically this should be equal to amount_required - amount_paid.

external_id
ID of the payment on paywall's system. Optional.

description
Payment description (max 128 chars).

fraud_status
Field representing the result of fraud check (only on supported backends).

fraud_message
Message provided along with the fraud status.

get_processor() → *getpaid.processor.BaseProcessor*
Returns the processor instance for the backend that was chosen for this Payment. By default it takes it from global backend registry and tries to import it when it's not there. You most probably don't want to mess with this.

get_unique_id() → str
Return unique identifier for this payment. Most paywalls call this "external id". Default: str(self.id) which is uuid4.

get_items() → List[getpaid.types.ItemInfo]
Some backends require the list of items to be added to Payment.

This method relays the call to Order. It is here simply because you can change the Order's fieldname when customizing Payment model. In that case you need to overwrite this method so that it properly returns a list.

get_form(**args*, ***kwargs*) → django.forms.forms.BaseForm
Interfaces processor's `get_form`.

Returns a Form to be used on intermediate page if the method returned by `get_redirect_method` is 'POST'.

get_template_names(*view=None*) → List[str]
Interfaces processor's `get_template_names`.

Used to get templates for intermediate page when `get_redirect_method` returns 'POST'.

handle_paywall_callback(*request*, ***kwargs*) → django.http.response.HttpResponse
Interfaces processor's `handle_paywall_callback`.

Called when 'PUSH' flow is used for a backend. In this scenario paywall will send a request to our server with information about the state of Payment. Broker can send several such requests during Payment's lifetime. Backend should analyze this request and return appropriate response that can be understood by paywall.

Parameters **request** – Request sent by paywall.

Returns HttpResponse instance

fetch_status() → getpaid.types.PaymentStatusResponse
Interfaces processor's `fetch_payment_status`.

Used during 'PULL' flow. Fetches status from paywall and proposes a callback depending on the response.

fetch_and_update_status() → getpaid.types.PaymentStatusResponse
Used during 'PULL' flow to automatically fetch and update Payment's status.

prepare_transaction(*request: Optional[django.http.request.HttpRequest] = None*, *view: Optional[django.views.generic.base.View] = None*, ***kwargs*) → django.http.response.HttpResponse
Interfaces processor's *prepare_transaction()*.

prepare_transaction_for_rest(*request: Optional[django.http.request.HttpRequest] = None*, *view: Optional[django.views.generic.base.View] = None*, ***kwargs*) → getpaid.types.RestfulResult
Helper function returning data as dict to better integrate with Django REST Framework.

confirm_prepared(***kwargs*) → None
Used to confirm that paywall registered POSTed form.

confirm_lock(*amount: Optional[Union[decimal.Decimal, float, int]] = None*, ***kwargs*) → None
Used to confirm that certain amount has been locked (pre-authed).

charge(*amount: Optional[Union[decimal.Decimal, float, int]] = None*, ***kwargs*) → getpaid.types.ChargeResponse
Interfaces processor's *charge()*.

confirm_charge_sent(***kwargs*) → None
Used during async charge cycle - after you send charge request, the confirmation will be sent to callback endpoint.

confirm_payment(*amount: Optional[Union[decimal.Decimal, float, int]] = None*, ***kwargs*) → None
Used when receiving callback confirmation.

mark_as_paid(***kwargs*) → None
Marks payment as fully paid if condition is met.

release_lock(***kwargs*) → decimal.Decimal
Interfaces processor's *charge()*.

start_refund(*amount: Optional[Union[decimal.Decimal, float, int]] = None*, ***kwargs*) → decimal.Decimal
Interfaces processor's *charge()*.

cancel_refund(***kwargs*) → bool
Interfaces processor's *charge()*.

confirm_refund(*amount: Optional[Union[decimal.Decimal, float, int]] = None*, ***kwargs*) → None
Used when receiving callback confirmation.

mark_as_refunded(***kwargs*) → None
Verify if refund was partial or full.

fail(***kwargs*) → None
Sets Payment as failed.

1.5 Creating payment plugins

In order to create a plugin for a payment broker, first you need to write a subclass of *BaseProcessor* named `PaymentProcessor` and place it in `processor.py` in your app.

The only method you have to provide is *prepare_transaction()* that needs to return a `HttpResponse` subclass (eg. HttpResponseRedirect or TemplateResponse). The use of all other methods depends directly on how the paywall operates.

To make your plugin available for the rest of the framework, you need to register it. The most convenient way to do so is `apps.py`:

```
from django.apps import AppConfig
```

(continues on next page)

(continued from previous page)

```
class MyPluginAppConfig(AppConfig):
    name = "getpaid_myplugin"
    verbose_name = "Some payment broker"

    def ready(self):
        from getpaid.registry import registry

        registry.register(self.module)
```

This way your plugin will be automatically registered after adding it to `INSTALLED_APPS`.

1.5.1 Detailed API

class getpaid.processor.**BaseProcessor**(*payment: django.db.models.base.Model*)

production_url = None
Base URL of production environment.

sandbox_url = None
Base URL of sandbox environment.

display_name = None
The name of the provider for the `choices`.

accepted_currencies = None
List of accepted currency codes (ISO 4217).

logo_url = None
Logo URL - can be used in templates.

ok_statuses = [200]
List of potentially successful HTTP status codes returned by paywall when creating payment

slug = None
For friendly urls

static get_our_baseurl(*request: Optional[django.http.request.HttpRequest] = None*, ***kwargs*) → str
Little helper function to get base url for our site. Note that this way 'https' is enforced on production environment.

prepare_form_data(*post_data: dict*, ***kwargs*) → Mapping[str, Any]
If backend support several modes of operation, POST should probably additionally calculate some sort of signature based on passed data.

get_form(*post_data: dict*, ***kwargs*) → django.forms.forms.BaseForm
(Optional) Used to get POST form for backends that use such flow.

abstract prepare_transaction(*request: django.http.request.HttpRequest*, *view: Optional[django.views.generic.base.View] = None*, ***kwargs*) → django.http.response.HttpResponse
Prepare Response for the view asking to prepare transaction.

Returns HttpResponse instance

handle_paywall_callback(*request: django.http.request.HttpRequest*, ***kwargs*) → django.http.response.HttpResponse
This method handles the callback from paywall for the purpose of asynchronously updating the payment status in our system.

Returns HttpResponse instance that will be presented as answer to the callback.

fetch_payment_status(***kwargs*) → getpaid.types.PaymentStatusResponse
Logic for checking payment status with paywall.

charge(*amount: Optional[Union[decimal.Decimal, float, int]] = None*, ***kwargs*) → getpaid.types.ChargeResponse
(Optional) Check if payment can be locked and call processor's method. This method is used eg. in flows that pre-authorize payment during order placement and charge money later.

release_lock(***kwargs*) → decimal.Decimal
(Optional) Release locked payment. This can happen if pre-authorized payment cannot be fullfilled (eg. the ordered product is no longer available for some reason). Returns released amount.

start_refund(*amount: Optional[Union[decimal.Decimal, float, int]] = None*, ***kwargs*) → decimal.Decimal
Refunds the given amount.

Returns the amount that is refunded.

cancel_refund(***kwargs*) → bool
Cancels started refund.

Returns True/False if the cancel succeeded.

1.6 Plugin registry

Plugin registry is a convenient way to handle multiple brokers that can support different currencies and provide different flows.

1.6.1 Internal API

class getpaid.registry.**PluginRegistry**

register(*module_or_proc*)
Register module containing PaymentProcessor class or a PaymentProcessor directly.

get_choices(*currency*)
Get CHOICES for plugins that support given currency.

get_backends(*currency*)
Get plugins that support given currency.

property urls
Provide URL structure for all registered plugins that have urls defined.

get_all_supported_currency_choices()
Get all currencies that are supported by at least one plugin, in CHOICES format.

1.7 Planned features

These features are planned for future versions (in no particular order):

- translations
- paywall communication log (all responses and callbacks)
- Subscriptions handling
- cookiecutter for plugins
- django-rest-framework helpers
- async/await support
- admin actions to PULL payment statuses

1.8 History

1.8.1 Version 2.2.0 (2020-05-03)

- Add template tag
- Add helper for REST integration

1.8.2 Version 2.1.0 (2020-04-30)

- Definitions for all internal data types and statuses
- Full type hinting
- Fixed bugs (thanks to Kacper Pikulski!)

1.8.3 Version 2.0.0 (2020-04-18)

- BREAKING: Complete redesign of internal APIs.
- Supports only Django 2.2+ and Python 3.6+
- Payment and Order became swappable models - like Django's User model
- Payment acts as customizable interface to PaymentProcessor instances (but be careful).
- Payment statuses guarded with django-fsm
- Broker plugins separated from main repo - easier updates.

1.8.4 Version 1.8.0 (2018-07-24)

- Updated project structure thanks to cookiecutter-djangopackage
- New plugin: pay_rest - New PayU API
- Updated following plugins: - payu - legacy API still works on new URL
- Dropped support for following plugins: - epaydk (API no longer functional) - moip (will be moved to separate package) - transferuj.pl (API no longer functional) - przelewy24.pl (API needs update, but no sandbox available anymore)
- Dropped support for Django <= 1.10
- Provide support for Django 2.0

1.8.5 Version 1.7.5

- Fixed przelewy24 params (py3 support)

1.8.6 Version 1.7.4

- Added default apps config getpaid.apps.Config
- Fixed and refactoring for utils.get_domain, build_absolute_uri, settings.GETPAID_SITE_DOMAIN
- Refactored register_to_payment
- Refactored build_absolute_uri
- Refactored and fixes in transferuj backend - payment.paid_on uses local TIMEZONE now as opposed to UTC - changed params - add post method to SuccessView and FailureView
- Added test models factories
- Dropped support for Django <=1.6

1.8.7 Version 1.7.3

- Refactored Dotpay
- Moved all existing tests to test_project and added more/refactored
- Fixed utils.import_module
- Fixed Payu and tests (py3 support)
- Updated docs

1.8.8 Version 1.7.2

- Updated coveragerc and travis.yml
- Added missing mgiration for Payment.status

1.8.9 Version 1.7.1

- Added coveragerc
- Updated README
- Added settings.GETPAID_ORDER_MODEL
- Added epay.dk support
- Added initial django migration

1.8.10 Version 1.7.0

- Refactoring to support for py3 (3.4)
- Change imports to be relative - fixes #43
- Add USD to supported currencies in Paymill backend (thanks lauris)
- Fix a few typos

1.8.11 Version 1.6.0

- Adding paymill backend
- PEP 8 improvements
- Adding support for django 1.5 in test project (+ tests)
- Fixed issue on *utils.import_name* to allow packages without parents
- Adding dependency to pytz for przelewy24 backend
- Refactoring of PayU backend (xml->txt api, better logging) and adding support for non-auto payment accepting

1.8.12 Version 1.5.1

- Fixing packaging that causes errors with package installation

1.8.13 Version 1.5.0

- Adding new backend - Przelewy24.pl (thanks to IssueStand.com funding)
- Fixing packaging package data (now using only MANIFEST.in)

1.8.14 Version 1.4.0

- Cleaned version 1.3 from minor issues before implementing new backends
- Brazilian backend moip
- Updated PL translation
- Added brazilian portuguese translation
- Storing payment external id and description in the database (warning: database migration needed!)
- Transferuj backend can now predefine interface language when redirecting
- POST method supported on redirect to payment

1.8.15 Version 1.3.0

- Logotypes support in new payment form
- Fixing packaging

1.8.16 Version 1.2

- Dotpay backend added
- Hooks for backends to accept email and user name
- Refactoring

1.8.17 Version 1.1

- PayU backend added
- Lots of documentation
- Refactoring

1.8.18 Version 1.0

- First stable version

CHAPTER

TWO

DEVELOPMENT TEAM

Project leader:

- Dominik Kozaczko <https://github.com/dekoza>

Original author:

- Krzysztof Dorosz <https://github.com/cypreess>.

Contributors:

- Paweł Bielecki <https://github.com/pawciobiel>
- Bernardo Pires Carneiro <https://github.com/bcarneiro>

Sponsors:

- ClearCode
- Sunscrapers

You are welcome to contribute to this project via github fork & pull request.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

A

AbstractOrder (*class in getpaid.models*), 9
AbstractPayment (*class in getpaid.models*), 10
accepted_currencies (*getpaid.processor.BaseProcessor attribute*), 13
amount_locked (*getpaid.models.AbstractPayment attribute*), 10
amount_paid (*getpaid.models.AbstractPayment attribute*), 10
amount_refunded (*getpaid.models.AbstractPayment attribute*), 10
amount_required (*getpaid.models.AbstractPayment attribute*), 10

B

backend (*getpaid.models.AbstractPayment attribute*), 10
BaseProcessor (*class in getpaid.processor*), 13

C

cancel_refund() (*getpaid.models.AbstractPayment method*), 12
cancel_refund() (*getpaid.processor.BaseProcessor method*), 14
charge() (*getpaid.models.AbstractPayment method*), 12
charge() (*getpaid.processor.BaseProcessor method*), 14
confirm_charge_sent() (*getpaid.models.AbstractPayment method*), 12
confirm_lock() (*getpaid.models.AbstractPayment method*), 12
confirm_payment() (*getpaid.models.AbstractPayment method*), 12
confirm_prepared() (*getpaid.models.AbstractPayment method*), 12
confirm_refund() (*getpaid.models.AbstractPayment method*), 12
created_on (*getpaid.models.AbstractPayment attribute*), 10
currency (*getpaid.models.AbstractPayment attribute*), 10

D

description (*getpaid.models.AbstractPayment attribute*), 10
display_name (*getpaid.processor.BaseProcessor attribute*), 13

E

external_id (*getpaid.models.AbstractPayment attribute*), 10

F

fail() (*getpaid.models.AbstractPayment method*), 12
fetch_and_update_status() (*getpaid.models.AbstractPayment method*), 11
fetch_payment_status() (*getpaid.processor.BaseProcessor method*), 14
fetch_status() (*getpaid.models.AbstractPayment method*), 11
fraud_message (*getpaid.models.AbstractPayment attribute*), 11
fraud_status (*getpaid.models.AbstractPayment attribute*), 11

G

get_absolute_url() (*getpaid.models.AbstractOrder method*), 9
get_all_supported_currency_choices() (*getpaid.registry.PluginRegistry method*), 14
get_backends() (*getpaid.registry.PluginRegistry method*), 14
get_buyer_info() (*getpaid.models.AbstractOrder method*), 10
get_choices() (*getpaid.registry.PluginRegistry method*), 14
get_description() (*getpaid.models.AbstractOrder method*), 10
get_form() (*getpaid.models.AbstractPayment method*), 11

get_form() (*getpaid.processor.BaseProcessor method*), 13
get_items() (*getpaid.models.AbstractOrder method*), 9
get_items() (*getpaid.models.AbstractPayment method*), 11
get_our_baseurl() (*getpaid.processor.BaseProcessor static method*), 13
get_processor() (*getpaid.models.AbstractPayment method*), 11
get_return_url() (*getpaid.models.AbstractOrder method*), 9
get_template_names() (*getpaid.models.AbstractPayment method*), 11
get_total_amount() (*getpaid.models.AbstractOrder method*), 10
get_unique_id() (*getpaid.models.AbstractPayment method*), 11

H

handle_paywall_callback() (*getpaid.models.AbstractPayment method*), 11
handle_paywall_callback() (*getpaid.processor.BaseProcessor method*), 13

I

id (*getpaid.models.AbstractPayment attribute*), 10
is_ready_for_payment() (*getpaid.models.AbstractOrder method*), 9

L

last_payment_on (*getpaid.models.AbstractPayment attribute*), 10
logo_url (*getpaid.processor.BaseProcessor attribute*), 13

M

mark_as_paid() (*getpaid.models.AbstractPayment method*), 12
mark_as_refunded() (*getpaid.models.AbstractPayment method*), 12

O

ok_statuses (*getpaid.processor.BaseProcessor attribute*), 13
order (*getpaid.models.AbstractPayment attribute*), 10

P

PluginRegistry (*class in getpaid.registry*), 14
prepare_form_data() (*getpaid.processor.BaseProcessor method*), 13
prepare_transaction() (*getpaid.models.AbstractPayment method*), 11
prepare_transaction() (*getpaid.processor.BaseProcessor method*), 13
prepare_transaction_for_rest() (*getpaid.models.AbstractPayment method*), 11
production_url (*getpaid.processor.BaseProcessor attribute*), 13

R

register() (*getpaid.registry.PluginRegistry method*), 14
release_lock() (*getpaid.models.AbstractPayment method*), 12
release_lock() (*getpaid.processor.BaseProcessor method*), 14

S

sandbox_url (*getpaid.processor.BaseProcessor attribute*), 13
slug (*getpaid.processor.BaseProcessor attribute*), 13
start_refund() (*getpaid.models.AbstractPayment method*), 12
start_refund() (*getpaid.processor.BaseProcessor method*), 14
status (*getpaid.models.AbstractPayment attribute*), 10

U

urls() (*getpaid.registry.PluginRegistry property*), 14