Pyramid Class-Based Views Documentation

Release 1.0

Adam Charnock

September 06, 2014

Contents

CHAPTER 1

Introduction

1.1 Class-based generic views?

So what are class-based generic views? A generic view is one which provides generic functionality. Implementing these types of views with classes allows the developer to customise this behaviour.

A simple example:

```
from pyramid_views import views
# An example TemplateView which simply renders the home template
class HomeView(views.TemplateView):
    template_name = 'templates/home.pt'
```

And to add a route:

```
config.add_route('home', '/', HomeView.as_view())
```

1.2 Why?

The motivation here is to allow for flexible views while removing the need for boiler-plate code.

1.3 Differences from Django

The functionality here is a direct port of Django's class-based views, so those docs are the recommended reference point.

However, there are a few differences to note:

- Any reference to the term `queryset` has changed to `query`. For example:
  - `get_queryset()` is now `get_query()`
  - The `queryset` attribute is now the `query` attribute
- The `__all__` value for the `fields` attribute is unsupported. Omit the `fields` attribute entirely for the same behaviour.

Additional functionality includes:

- The `MacroMixin` class, for passing macros to chameleon templates via the `macro_names` attribute.
- The `DbSessionMixin` class makes the database session available via `self.db_session`

- The `UpdateView` class supports the `partial_updates` option. When `True`, this will only update fields present in the request
- Form-based can take the `endpoint` parameter. When `True`, the view will behave like an API endpoint rather than a user-facing page (errors encoded as JSON, 200 response codes when `success_uri` not present).

CHAPTER 2

Future Development

Potential future development as follows:

- Pyramid access control list support
- Implementation of the (currently unimplemented) date views

CHAPTER 3

API Reference

3.1 View

class pyramid_views.views.base.**View**(***kwargs*)

Intentionally simple parent class for all views. Only implements dispatch-by-method and simple sanity checking.

@type request: pyramid.request.Request @type args: list @type kwargs: dict

get(*self*, *request*, **args*, ***kwargs*)

post(*self*, *request*, **args*, ***kwargs*)

put(*self*, *request*, **args*, ***kwargs*)

delete(*self*, *request*, **args*, ***kwargs*)

classmethod as_view(***initkwargs*)

Main entry point for a request-response process.

dispatch(*request*, **args*, ***kwargs*)

http_method_not_allowed(*request*, **args*, ***kwargs*)

options(*request*, **args*, ***kwargs*)

Handles responding to requests for the OPTIONS HTTP verb.

args = None

http_method_names = ['get', 'post', 'put', 'patch', 'delete', 'head', 'options', 'trace']

kwargs = None

request = None

3.2 Redirect View

class pyramid_views.views.base.**RedirectView**(***kwargs*)

A view that provides a redirect on any GET request.

permanent = True

url = None

pattern_name = None

query_string = False

get_redirect_url(**elements*, ***kw*)
Return the URL redirect to. Keyword arguments from the URL pattern match generating the redirect request are provided as kwargs to this method.

3.3 Template View

class pyramid_views.views.base.**TemplateView**(***kwargs*)
A view that renders a template. This view will also pass into the context any keyword arguments passed by the url conf.

template_name = None

content_type = None

get_context_data(***kwargs*)

macro_names = None

get_macro_names()
Return a directory of macro names.

Values should be template paths, and keys will be used as the lookup key in the template. Eg. `macros.<key>.<macro>`.

get(*request*, **args*, ***kwargs*)

3.4 List View

class pyramid_views.views.list.**ListView**(***kwargs*)
Render some list of objects, set by `self.model` or `self.query`. `self.query` can actually be any iterable of items, not just a query.

model = None

query = None

get_query()
Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

template_name = None

template_extension = '.pt'

content_type = None

template_name_suffix = '_list'

get_context_data(***kwargs*)
Get the context for this view.

get_context_object_name(*object_list*)
Get the name of the item to be used in the context.

allow_empty = True

get_allow_empty()
Returns `True` if the view should display empty lists, and `False` if a 404 should be raised instead.

db_session

macro_names = None

get_macro_names()
Return a directory of macro names.

Values should be template paths, and keys will be used as the lookup key in the template. Eg. `macros.<key>.<macro>`.

paginate_by = None

get_paginate_by(*query*)
Get the number of items to paginate by, or `None` for no pagination.

paginate_orphans = 0

get_paginate_orphans()
Returns the maximum number of orphans extend the last page by when paginating.

page_kwarg = 'page'

get(*request*, **args*, ***kwargs*)

3.5 Detail View

class pyramid_views.views.detail.**DetailView**(***kwargs*)
Render a "detail" view of an object.

By default this is a model instance looked up from `self.query`, but the view will support display of *any* object by overriding `self.get_object()`.

model = None

query = None

get_query()
Return the *Query* that will be used to look up the object.

Note that this method is called by the default implementation of *get_object* and may not be called if *get_object* is overriden.

get_object(*query=None*)
Returns the object the view is displaying.

By default this requires *self.query* and a *pk* or *slug* argument in the URLconf, but subclasses can override this to return any object.

slug_field = u'slug'

get_slug_field()
Get the name of a slug field to be used to look up by slug.

slug_url_kwarg = u'slug'

pk_url_kwarg = u'pk'

template_name = None

template_extension = u'.pt'

content_type = None

template_name_suffix = u'_detail'

get_context_data(***kwargs*)

get_context_object_name(*obj*)
Get the name to use for the object.

db_session

macro_names = None

get_macro_names()
Return a directory of macro names.

Values should be template paths, and keys will be used as the lookup key in the template. Eg. `macros.<key>.<macro>`.

get(*request*, **args*, ***kwargs*)

3.6 Form View

class pyramid_views.views.edit.**FormView**(***kwargs*)
A view for displaying a form, and rendering a template response.

success_url = None

form_class = None

get_form_class()
Returns the form class to use in this view

get_form(*form_class*)
Returns an instance of the form to be used in this view.

form_invalid(*form*)
If the form is invalid, re-render the context data with the data-filled form and errors.

form_valid(*form*)
If the form is valid, redirect to the supplied URL.

Note: Override this method if you wish to perform an action on successful form submission (Eg: For a contact form you may wish to send an email).

---

prefix = None

get_prefix()
Returns the prefix to use for forms on this view

initial = {}

get_initial()
Returns the initial data to use for forms on this view.

get_form_kwargs()
Returns the keyword arguments for instantiating the form.

template_name = None

content_type = None

get_context_data(***kwargs*)

get(*request*, **args*, ***kwargs*)

post(*request*, **args*, ***kwargs*)
Handles POST requests, instantiating a form instance with the passed POST variables and then checked for validity.

3.7 Create View

class pyramid_views.views.edit.**CreateView**(***kwargs*)
View for creating a new object instance, with a response rendered by template.

success_url = None
The URL to redirect to upon successful object creation.

fields = None
Fields which should be presented to the user. If `None`, all fields will be available with certain field exclusion rules.

Important: It is highly recommended you specify this field in order to prevent fields becoming unintentionally presented to the user.

---

model = None
The model of which an instance will be created.

query = None

get_query()
Return the *Query* that will be used to look up the object.

Note that this method is called by the default implementation of *get_object* and may not be called if *get_object* is overriden.

get_object(*query=None*)
Returns the object the view is displaying.

By default this requires *self.query* and a *pk* or *slug* argument in the URLconf, but subclasses can override this to return any object.

slug_field = u'slug'

get_slug_field()
Get the name of a slug field to be used to look up by slug.

slug_url_kwarg = u'slug'

pk_url_kwarg = u'pk'

form_class = None

get_form_class()
Returns the form class to use in this view.

get_form(*form_class*)
Returns an instance of the form to be used in this view.

form_invalid(*form*)
If the form is invalid, re-render the context data with the data-filled form and errors.

form_valid(*form*)
If the form is valid, save the associated model.

prefix = None

get_prefix()
Returns the prefix to use for forms on this view

initial = {}

get_initial()
Returns the initial data to use for forms on this view.

get_form_kwargs()
Returns the keyword arguments for instantiating the form.

template_name = None

content_type = None

get_context_data(***kwargs*)

macro_names = None

get_macro_names()
Return a directory of macro names.

Values should be template paths, and keys will be used as the lookup key in the template. Eg. `macros.<key>.<macro>`.

get(*request*, **args*, ***kwargs*)

post(*request*, **args*, ***kwargs*)

3.8 Update View

class pyramid_views.views.edit.**UpdateView**(***kwargs*)
View for updating an object, with a response rendered by template.

success_url = None
The URL to redirect to upon successful update.

fields = None
Fields which should be presented to the user. If `None`, all fields will be available with certain field exclusion rules.

Important: It is highly recommended you specify this field in order to prevent fields becoming unintentionally presented to the user.

---

model = None
The model of which an instance will be updated.

query = None
Limit updating to only objects provided by `query`. If you specify this then you can omit `model`.

get_query()
Return the *Query* that will be used to look up the object.

Note that this method is called by the default implementation of *get_object* and may not be called if *get_object* is overriden.

get_object(*query=None*)
Returns the object the view is displaying.

By default this requires *self.query* and a *pk* or *slug* argument in the URLconf, but subclasses can override this to return any object.

slug_field = u'slug'

get_slug_field()
Get the name of a slug field to be used to look up by slug.

slug_url_kwarg = u'slug'

pk_url_kwarg = u'pk'

form_class = None

get_form_class()
Returns the form class to use in this view.

get_form(*form_class*)
Returns an instance of the form to be used in this view.

form_invalid(*form*)
If the form is invalid, re-render the context data with the data-filled form and errors.

form_valid(*form*)
If the form is valid, save the associated model.

prefix = None

get_prefix()
Returns the prefix to use for forms on this view

initial = {}

get_initial()
Returns the initial data to use for forms on this view.

get_form_kwargs()
Returns the keyword arguments for instantiating the form.

template_name = None

content_type = None

get_context_data(***kwargs*)

macro_names = None

get_macro_names()
Return a directory of macro names.

Values should be template paths, and keys will be used as the lookup key in the template. Eg. `macros.<key>.<macro>`.

get(*request*, **args*, ***kwargs*)

post(*request*, **args*, ***kwargs*)

3.9 Delete View

class pyramid_views.views.edit.**DeleteView**(***kwargs*)
View for deleting an object retrieved with *self.get_object()*, with a response rendered by template.

success_url = None
The URL to redirect to upon successful deletion.

model = None
The model of which an instance will be deleted.

query = None
Limit deletion to only objects provided by `query`. If you specify this then you can omit `model`.

get_query()
Return the *Query* that will be used to look up the object.

Note that this method is called by the default implementation of *get_object* and may not be called if *get_object* is overriden.

get_object(*query=None*)
Returns the object the view is displaying.

By default this requires *self.query* and a *pk* or *slug* argument in the URLconf, but subclasses can override this to return any object.

slug_field = u'slug'

get_slug_field()
Get the name of a slug field to be used to look up by slug.

slug_url_kwarg = u'slug'

pk_url_kwarg = u'pk'

template_name = None

content_type = None

get_context_data(***kwargs*)

macro_names = None

get_macro_names()
Return a directory of macro names.

Values should be template paths, and keys will be used as the lookup key in the template. Eg. `macros.<key>.<macro>`.

get(*request*, **args*, ***kwargs*)

post(*request*, **args*, ***kwargs*)

This site documents pyramid-views, a Pyramid port of Django's class-based generic views.

Note: Note that the most complete documentation will be Django's class-based view reference documentation. You may also find the Django's class-based view introductory documentation useful.

See also:

- Django's class-based view introductory documentation
- Django's class-based view reference documentation
- GitHub
- PyPi

p