django-smarter Documentation

Release 1.0 beta

Alexey Kinyov

September 28, 2016

Contents

Another approach for declarative style generic views for Django. I beleive, it's a bit smarter :)

CHAPTER 1

Overview

So many times we have to write:

```
@login_required
def edit_post(request, pk):
    post = get_object_or_404(Post, pk=pk)
    if request.method == 'POST':
        form = EditPostForm(request.POST, instance=post)
        if form.is_valid():
            post = form.save()
            return redirect(post.get_absolute_url())
    else:
        form = EditPostForm()
    return render(request, 'edit_post.html', {'form': form})
```

Right? Well, it's ok to write some reusable helpers for such repeatable views, but when we don't need sophisticated ones here we go:

```
class PostViews(smarter.GenericViews):
    model = Post
    options = {
        'add': {
            'form': NewPostForm,
            'decorators': (login_required,)
        },
        'edit': {
            'form': EditPostForm,
            'decorators': (login_required,)
        },
        'remove': {
            'decorators': (login_required,)
        }
    }
```

That's it.

CHAPTER 2

Changes in v1.0

API is finally and completely changed since v0.6 release.

We've made a "quantum jump" by breaking old-and-not-so-good API to new one - solid and nice. Hope you'll like it.

Here are some hints that may help you with migration. I'm actually successfully migrated my real-production project, so the hints are based on "real-battle" example.

CHAPTER 3

Contributors

- Fabio Santos
- Sameer Al-Sakran

Thank you, comrades! :)

CHAPTER 4

Installation

Requirements:

- Django >= 1.4

Installation:

```
pip install django-smarter
```

You *may* add `smarter` to your `INSTALLED_APPS` to get default templates and tests, but you *don't have to*:

```
INSTALLED_APPS = (
    # ...
    'smarter',
    # ...
)
```

Then you should define your views and include them in URLs, see *Getting started* section below.

CHAPTER 5

Getting started

5.1 Create your models

Let's define a simple model:

```
class Page(models.Model):
    owner = models.ForeignKey('auth.User')
    title = models.CharField(max_length=100)
    text = models.TextField()

    def __unicode__(self):
        return self.title
```

5.2 Register views

Now you can add generic views for the model.

In your *urls.py*:

```
import smarter
from myapp.models import Page

site = smarter.Site()
site.register(smarter.GenericViews, Page)

urlpatterns = patterns('',
    url(r'^', include(site.urls)),

    # other urls ...
)
```

This code creates generic views for `Page` model, accessed by urls:

- /page/
- /page/add/
- /page/<pk>/
- /page/<pk>/edit/
- /page/<pk>/remove/

5.3 Customize views

Subclass from `smarter.GenericViews` and set custom options and/or override methods.

```
from django.contrib.auth.decorators import login_required
import smarter
from .models import Page

class PageViews(smarter.GenericViews):
    model = Page

    options = {
        'add': {
            'decorators': (login_required,)
            'exclude': ('owner',)
        },
    }

    def add__save(self, request, form, **kwargs):
        obj = form.save(commit=False)
        obj.owner = request.user
        obj.save()
        return obj
```

And don't forget to register new views in *urls.py*:

```
import smarter
from myapp.views import PageViews

site = smarter.Site()
site.register(PageViews) # model argument is not required as model is already set in PageViews

urlpatterns = patterns('',
    url(r'^', include(site.urls)),
)
```

5.4 Customize templates

In the example above each URL by default to template.

URL	Template	Context
/page/	myapp/page/index.html	{{ objects_list }}
/page/add/	myapp/page/add.html	{{ obj }}, {{ form }}
/page/<pk>/	myapp/page/details.html	{{ obj }}
/page/<pk>/edit/	myapp/page/edit.html	{{ obj }}, {{ form }}
/page/<pk>/remove/	myapp/page/remove.html	{{ obj }}

Default template search paths are:

```
('%(app)s/%(model)s/%(action)s.html',
 '%(app)s/%(model)s/%(action)s.ajax.html',
 'smarter/%(action)s.html',
 'smarter/_form.html',
 'smarter/_ajax.html',)
```

So, you have some easy way options:

1. you may override matching templates
2. you may set 'template' key in `PageViews.options` for each action
3. you may override default search paths by settings new `PageViews.defaults` (read *Options* section for details)

5.5 Singleton Site

A very special instance of *smarter.Site* is in the smarter module. It allows you to register your applications' views outside your urls.py file, and works well with *autodiscover()*.

Here is smarter_views.py in your app:

```
from smarter import site, GenericViews
from models import Model

class Views(GenericViews):
    model = Model

    # ...

site.register(Views)
```

... And urls.py:

```
from django.conf.urls import patterns, include, url
import smarter

smarter.autodiscover()
urlpatterns = patterns('',
    url(r'^', include(smarter.site.urls)),
)
```

This is mostly recommended for non-reusable applications local to your Django project.

CHAPTER 6

API reference

6.1 Actions

Actions are actually "ids" for views. Well, each action has id like 'add', 'edit', 'bind-to-user' and is mapped to view method with underscores instead of '-': *add*, *edit*, *bind_to_user*.

In `smarter.GenericViews` class such actions are defined by default:

Action	URL	View method	Named URL
index	/	index(`request`)	[prefix]-[model]-index
add	/add/	add(`request`)	[prefix]-[model]-add
details	/<pk>/	details(`request, pk`)	[prefix]-[model]-details
edit	/<pk>/edit/	edit(`request, pk`)	[prefix]-[model]-edit
remove	/<pk>/remove/	remove(`request, pk`)	[prefix]-[model]-remove

What is **[prefix]**? Prefix is defined for `smarter.Site` instance:

```
site = smarter.Site(prefix='myapp')
site.register(PageViews)
# ...
```

So, it **can be empty** and URL names without prefix are defined as *[model]-index*. Please, read *Reversing urls* section for more details.

6.2 Options

Options is a `GenericViews.options` dict, class property, it contains actions names as keys and actions parameters as values. Parameters structure is:

```
{
    'url':          <string for url pattern>,
    'form':         <form class>,
    'decorators':   <tuple/list of decorators>,
    'fields':       <tuple/list of form fields>,
    'exclude':      <tuple/list of excluded form fields>,
    'initial':      <tuple/list of form fields initialized by request.GET>,
    'permissions':  <tuple/list of required permissions>,
    'widgets':      <dict for widgets overrides>,
    'help_text':    <dict for help texts overrides>,
    'required':     <dict for required fields overrides>,
    'template':     <string template name>,
```

```
    'redirect':     <string or callable returning redirect path>
}
```

Every key here is optional. So, here's how options can be defined for views:

```
import smarter

class Views(smarter.GenericViews):
    model = <model>

    defaults = <default parameters>

    options = {
        '<action 1>': <parameters 1>,
        '<action 2>': <parameters 2>
    }
```

And here's `GenericViews.defaults` class attribute:

```
defaults = {
    'initial': None,
    'form': ModelForm,
    'exclude': None,
    'fields': None,
    'labels': None,
    'widgets': None,
    'required': None,
    'help_text': None,
    'next': None,
    'template': (
        '%(app)s/%(model)s/%(action)s.html',
        '%(app)s/%(model)s/%(action)s.ajax.html',
        'smarter/%(action)s.html',
        'smarter/_form.html',
        'smarter/_ajax.html',),
    'decorators': None,
    'permissions': None,
}
```

When option value can't be found in options dict for action it's searched in *GenericViews.defaults*. Note, that defaults are applied to **all actions**.

6.3 Action names and URLs

Actions are named so they can be mapped to views methods and they should not override reserved attributes and methods, so they:

1. **must contain only** latin symbols and '_' or '-', **no spaces**
2. **can't** be in this list: 'model', 'defaults', 'options', 'deny'
3. **can't** start with '-', '_' or 'get_'
4. **can't** contain '__'

Sure, you'll get an exception if something goes wrong with that. We're following *'errors should never pass silently'* here.

And here's how URLs for default views are defined:

```
{
    'index': {
        'url': r'',
    },
    'details': {
        'url': r'(?P<pk>\d+)/',
    },
    'add': {
        'url': r'add/',
    },
    'edit': {
        'url': r'(?P<pk>\d+)/edit/',
    },
    'remove': {
        'url': r'(?P<pk>\d+)/remove/',
    }
}
```

6.4 smarter.Site

Site(prefix=None, delim='-')
- constructor

register(views, model=None, base_url=None, prefix=None)
- method to add your views for model

urls
- property, returns URLs sequence for all registered views that can be included in *urlpatterns*

autodiscover
- method which goes over *settings.INSTALLED_APPS* and looks for apps with *smarter_views* modules, which it imports, so they can register their views.

6.4.1 Site

Constructor gets two keyword arguments:

1. *prefix=None*, for prefixing URL names for views registered with site object, like '**%(prefix)s**-%(model)s-%(action)s'. If prefix if empty, URLs are named without prefix, like '%(model)s-%(action)s'.
2. *delim='-'*, delimiter for URL names, can be '-', '_' or empty string. URL names are composed with specified delimiter and with uderscore it would be like '%(prefix)s_%(model)s_%(action)s'.

6.4.2 Site.register

This method gets 1 required argument for views class and optional keyword arguments:

1. *model=None*, model class for views. This argument is required if views class doesn't have 'model' property.
2. *base_url=None*, base URL for views. If empty, then lower-case model name is used, so base URL becomes '%(model)s/'.

3. *prefix=None*, prefix for URL names. If empty, then lower-case model name is used.

6.5 smarter.GenericViews

model
- class property, model class for views

defaults
- class property, dict with default options applied to all actions until being overriden by *options*

options
- class property, dict for views configration, each key corresponds to single action like 'add', 'edit', 'remove' etc.

deny(`request, message=None`)
- method, is called when action is not permitted for user, raises `PermissionDenied` exception or can return `HttpResponse` object for redirecting or rendering some page

get_url(`action, *args, **kwargs`)
- method, returns url for given action name

get_form(`request, **kwargs`)
- method, returns form for request

get_object(`request, **kwargs`)
- method, returns single object for request

get_objects_list(`request, **kwargs`)
- method, returns objects for request

get_template(`request_or_action`)
- method, returns template name or sequence of template names by action name or per-request

get_param(`self, request_or_action, name, default=None`)
- method, returns option parameter by name for action or per-request

get_initial(`self, request`)
- method, returns form initial data per-request

<action>(`request, **kwargs`)
- method, 1st (starting) handler in default pipeline

<action>__perm(`request, **kwargs`)
- method, 2nd handler in default pipeline, checks extended permissions, e.g. per-object permissions (basic checks are handler separatelly)

<action>__form(`request, **kwargs`)
- method, 3rd handler in default pipeline, manages form processing

<action>__save(`request, form, **kwargs`)
- method, called from **<action>__form** when form is ready to save, saves the form and returns saved instance

<action>__post(`request, **kwargs`)
- method, 4th handler in default pipeline for post-processing: save messages, extend render context, etc.

<action>__done(`request, **kwargs`)
- method, 5th (last) view handler in default pipeline, performs render or redirect

6.6 Pipeline

Each action like 'add', 'edit' or 'remove' is a **pipeline**: a sequence (list) of methods called one after another. A result of each method is passed to the next one.

The result is either **None** or **dict** or **HttpResponse** object:

1. **None** - result from previous pipeline method is used for next one,
2. **dict** - result is passed to next pipeline method,
3. **HttpResponse** - returned immidiately as view response.

For example, 'edit' action pipeline is: 'edit' -> 'edit__perm' -> 'edit__form' -> 'edit__post' -> 'edit__done'.

Note about **__perm** step. Basic permissions are checked **before** pipeline start view (e.g 'edit'), as if view were decorated with `permission_required` decorator. Actualy we're not using decorator, because we need to call our custom `deny()` method if permissions are not sufficient, but it's not the key. The key is **you don't need to check basic permissions in custom __perm method, it's necessary for per-object permissions checks.**

Method	Parameters	Result
edit	`request, **kwargs` 'pk'	`{'obj': obj, 'form': {'instance': obj}}`
edit__perm	`request, **kwargs` 'obj', 'form'	pass (`None`) or `PermissionDenied` exception
edit__form	`request, **kwargs` 'obj', 'form'	`{'form': form, 'obj': obj, 'form_saved': True}` - form successfully saved `{'form': form, 'obj': obj}` - first open or form contains errors
edit__post	`request, **kwargs` 'obj', 'form', 'form_saved'	pass (`None`) by default
edit__done	`request, **kwargs` 'obj', 'form', 'form_saved'	render template or redirect to `obj.get_absolute_url()`

Note, that in general you won't need to redefine pipeline methods, as in many cases custom behavior can be reached with declarative style using **options**. If you're going too far with overriding views, that may mean you'd better write

some views from scratch separate from “smarter”.

6.7 Reversing URLs

Every action mapped to named URL. Names are composed as:

```
[site prefix][delimiter][views prefix][delimiter][action]
```

Where:

- **site prefix** is ‘prefix’ parameter in *smarter.Site* constructor
- **delimiter** is ‘delim’ paratemer in *smarter.Site* constructor
- **views prefix** is ‘prefix’ parameter in *Site.register* method

So, in *Getting started* example named URLs are ‘page-add’, ‘page-edit’, ‘page-remove’, etc., as we don’t provide any custom prefixes and delimiter is ‘-‘ by default.

CHAPTER 7

Pipeline example

For deeper understanding here's an example of custom pipeline for 'edit' action. It's not actually a **recommended** way, as we can reach the same effect without overriding `edit` method by defining `options['edit']['initial']`, but it illustrates the principle of pipeline.

```
import smarter

class PageViews(smarter.GenericViews):
    model = Page

    def edit(request, pk=None):
        # Custom initial title
        initial = {'title': request.GET.get('title': '')}
        return {
            'obj': self.get_object(request, pk=pk),
            'form' {'initial': initial, 'instance': obj}
        }

    def edit__perm(request, **kwargs):
        # Custom permission check
        if kwargs['obj'].owner != request.user:
            return self.deny(request)

    def edit__form(request, **kwargs):
        # Actually, nothing custom here, it's totally generic:
        # we should validate & save form and then return dict
        # with 'form_saved' set to True if it's ok.
        kwargs['form'] = self.get_form(request, **kwargs)
        if kwargs['form'].is_valid():
            kwargs['obj'] = self.edit__save(request, **kwargs)
            kwargs['form_saved'] = True
        return kwargs

    def edit__done(request, obj=None, form=None, form_saved=None):
        # Custom redirect to pages index on success
        if form_saved:
            # Success, redirecting!
            return redirect(self.get_url('index'))
        else:
            # Start edit or form has errors
            return render(request, self.get_template(request),
                          {'obj': obj, 'form': form})
```

CHAPTER 8

Complete example

You may look at complete example source here:
https://github.com/05bit/django-smarter/tree/master/example

CHAPTER 9

License

Copyright (c) 2013, Alexey Kinyov <rudy@05bit.com> Licensed under BSD, see LICENSE for more details.

9.1 Hints for updating app from older smarter

9.1.1 Base API changes

- name_prefix -> prefix
- SmarterSite -> Site
- smarter.views.GenericViews -> smarter.GenericViews
- register(views_or_model, generic_views=None) -> register(views, model)

9.1.2 URLs paths

Define 'url' for custom actions (search for `urls_custom`).

9.1.3 URLs names

- prefix='([^']+)-' -> prefix='1'

Template paths:

- Move templates to new paths:

```
('%(app)s/%(model)s/%(action)s.html',
 '%(app)s/%(model)s/%(action)s.ajax.html',
 'smarter/%(action)s.html',
 'smarter/_form.html',
 'smarter/_ajax.html',)
```

- or redefine 'template' in defaults, e.g:

```
('%(app)s/%(model)s_%(action)s.html',
 '%(model)s_%(action)s.html',
 'smarter/%(action)s.html')
```

9.1.4 Decorators

Now defined in options as 'decorators' tuple/list, no 'method_decorator' needed.

9.1.5 AJAX

Define 'ajax' handler in options.

9.1.6 Permissions

`GenericViews.check_permissions()` is not called anymore, use 'permissions' options and `GenericViews.{action}__perm` methods.

9.1.7 Form save

`GenericViews.save_form()` is not called anymore, use `GenericViews.{action}__save` methods.

9.1.8 Views

- {action}_view -> {action}
- {action} method should return dict instead if `HttpResponse`
- no self.process_form() - it's not needed anymore
- `update_context` is not called anymore, use `{action}__post` methods
- no `render_to_response` method anymore, use Django `render` shortcut with `GenericViews.get_templates` method
- `get_object` and `get_objects_list` require request object as first argument
- `deny` method requires request object as argument
- form_params_[action] -> [action]'s 'form' in result dict

CHAPTER 10

Indices and tables

- genindex
- modindex
- search