Client Admin Documentation

Release 1.0.12

Kyle Rimkus

Oct 07, 2017

Contents

Client Admin is an open source project developed by Concentric Sky that enhances Django Admin by providing organization tools and features that make the interface more appropriate for clients. Some of the code started as Admin Tools, although the theming engine has been removed and many features have been added.

CHAPTER 1

Table of Contents

Installation

pip install git+https://github.com/concentricsky/django-client-admin.git

Include Client Admin in your settings.py. It should be listed before the django.contrib.admin module so that client_admin can override the default Django admin templates.

```
INSTALLED_APPS = [
    'client_admin',

    ...

]
```

Include Client Admin urls in your urls.py at the same path as normal contrib.admin

```
urlpatterns = patterns('',

    ...

    url(r'^admin/', include('client_admin.urls')),
    url(r'^admin/', include('admin.site.urls')),

    ...

)
```

Import and inherit from ClientModelAdmin and Inline classes.

```
from client_admin.admin import ClientModelAdmin, GroupedInline, TabularInline,␣
↪StackedInline


class BindingInline(GroupedInline):
```

```
    model = Binding


class RetailerInline(TabularInline):
    model = Retailer


class AuthorInline(StackedInline):
    model = Author


class BookAdmin(ClientModelAdmin):
    inlines = [PhoneNumberInline, RetailerInline, MailingAddressInline]

...
```

Customization

Branding Client Admin for each project can be done by providing a replacement header menu logo. In the project's static folder, create a client_admin/images/h1_logo_bg.png that is 40px tall and no more than 400px wide.

Best Practices

- **Create a 'staff' user that belongs to a 'staff' group with limited permissions.**
  - **The staff group should have access to:**
    - Editing and creating all site-specific structured content
    - Deleting any structured content that is used as an inline
  - **The staff group should _not_ have access to:**
    - Deleting any structured content that would cascade delete. Instead, have them use an 'is_active' flag to remove content from the front end.
    - Administration apps like Auth and Sites

Features

Default Features

- Provides a new style for the Django Admin interface.
- Provides a dashboard with draggable widgets.
- Creates a menu persistent on all admin pages.
- Provides a system of admin bookmarks that allow users to quickly star favorite pages and have them available from the menu.
- Provides a ClientModelAdmin class to inherit from that improves default widgets and settings.
- Provides an additional inline type, Grouped, that acts much like a Stacked inline but floats each field group instead of clearing them.

- Allows admin templates to extend Jinja2 templates. Assuming certain blocks are present in your template, this means the admin interface could inherit a header and footer from the front-end templates.

Additional Features

- Provides nested inline formsets for ModelAdmin classes.
- Adds an advanced search form to change list views.
- Provides an improved generic-foreignkey widget.
- Provides an improved Raw ID foreignkey widget that displays unicode instead of the object's pk.
- Includes revision history and deleted object recovery via django-reversion

Dashboard Widgets

App List

The App List module shows a list of models with links to add objects or show the change-list view for the model. The list can be limited to either show or exclude certain apps or models. By default, two instances of App Lists are included on the dashboard, one showing all models that aren't part of the django.contrib app and one showing only those from django.contrib.

Link List

The Link List module displays a list of hrefs that can either be external urls or internal urls using the reverse() function.

Memcached Status

If memcached is being used on the server, then this module displays detailed statistics from each memcached instance.

Recent Activity

The Recent Activity module lists all LogEntry objects, grouped by which user generated them. Normally, the LogEntry objects are only created from admin activity, but this can also be used to display activity generated from the front-end of the site if those views create LogEntry objects using the log_action() function.

Sitemap

The sitemap module allows listing models in a hierarchy similar to a front-end site menu. This allows admin users to find content in the same logical place that a front-end user would see. By default, Client Admin will build the hierarchy based on a menu from Sky CMS, currently a private library. This code will eventually be pulled out of Client Admin and only included in Sky CMS.

Recursive Inlines

Basic Django admin allows for a model to be edited inline with a related model, but it is limited to one level of nesting. Client Admin provides several classes that allow inlines to be nested recursively.

Example:

```
class ChapterInline(client_admin.TabularInline):
    model = Chapter

class BookInline(client_admin.StackedInline):
    model = Book
    inlines = [ChapterInline]

class AuthorAdmin(client_admin.ClientModelAdmin):
    model = Author
    inlines = [BookInline, AwardInline]
```

By default *RecursiveInlinesModelAdmin* is inherited by *ClientModelAdmin*. It is possible to inherit from and use the *RecursiveInlinesModelAdmin* class on its own. Normally, only *ClientModelAdmin* and the inline classes from Client Admin would be used.

Advanced Search

If search fields are provided to a ClientModelAdmin class, they will automatically be included in an advanced search form that gets appended to the change list template but hidden by default. A link will be added just below the normal search form in the sidebar to expand the advanced search form. All search fields will be listed, including properties of foreignkey and many-to-many fields. The form field labels are automatically generated for related fields as <*model*> <*field*> but can be overridden using a property on the admin class of *advanced_search_titles*.

Example:

```
class AwardInline(admin.StackedInline):
    model = Award

class BookInline(StackedInline):
    model = Book

class AuthorAdmin(client_admin.RecursiveInlinesModelAdmin):
    model = Author
    search_fields = ('name', 'book_set__title', 'award_set__name')
    advanced_search_titles = {
        'book_set__title': 'Titles',
        'award_set__name': 'Awards'
    }
```

What's in the works?

Structure

- Responsive templates
- All views re-written as class-based
- Abstracting list views as reusable ModelTables

Features

- Create a Jinja2 helper that would print the admin menu bar on front-end templates. This would most likely be inserted dynamically and position absolutely to maintain the existing front-end structure.
- Create Jinja2 helpers that insert links to corresponding admin pages for content displayed in front-end templates. Again, these would most likely be inserted dynamically and position absolutely to maintain the existing front-end structure.
- Provide a modal interface to edit WYSIWYG fields (currently CKEditor) directly from the front end.
- Support including foreign keys and many-to-many relationships as inlines on a change form. This would mean the original field would be excluded in the main form, and the inverse inlines would be saved first, with their resulting PKs saved as the value for the corresponding relationship on the main form.
- Support for nested inlines using AJAX submissions of inline forms instead of nested formsets.