Django Candy

Bharat Chauhan

Sep 02, 2021

CONTENTS

django-candy is a modern, single-page-app admin app for Django. The frontend UI is written in React JS.

> **Attention:** Django Candy is currently a **pre-alpha project**. Most of the things don't work. However, if you're curious, you can still try it out.

CHAPTER

ONE

PROJECT LINKS

- Current version: 0.7.2 (*Release notes*)
- Source code (Github)
- Frontend source code (Github)

CHAPTER

TWO

TABLE OF CONTENTS

2.1 Installation

Attention: Django Candy is currently a **pre-alpha project**. Most of the things don't work. However, if you're curious, you can still try it out.

Install using pip:

```
$ pip install django-candy
```

Update your project's settings:

```
# settings.py
INSTALLED_APPS = [
    # ...
    'django_candy'
]
```

Update your project's urls:

```
# urls.py
urlpatterns = [
    # ...
    path('candy/', include('django_candy.urls')),
]
```

Run the server and visit `http://127.0.0.1:8000/candy/` from your browser to see the admin in action.

For logging in, you need to create a user using Django's `createsuperuser` command.

Next, go to *Quickstart* page for instructions about using the admin site.

2.2 Quickstart

Candy provides an API quite similar to that of Django admin's.

Note: As of version 0.7.2, `django-candy` is not a drop-in replacement for Django admin. See *Differences from Django admin* for more.

2.2.1 Register your models

```
# admin.py

from django_candy import admin
from myapp.models import MyModel

admin.site.register(MyModel)
```

If you reload the Candy admin site, you should find your model appear in the side menu.

2.2.2 `ModelAdmin` class

Candy also provides a *`ModelAdmin`* class similar to that of Django admin.

To choose which fields to display in the list page, you can do this:

```
# admin.py

from django_candy import admin
from myapp.models import MyModel

class MyModel(admin.ModelAdmin):
    list_display = ['field_1', 'field_2', 'etc']

admin.site.register(MyModel, MyModelAdmin)
```

As of version 0.7.2, only the list page works (and partially so).

2.2.3 Next steps

As mentioned earlier, `django-candy` is not a drop-in replacement for Django's default admin. It's a good idea to familiarise yourself with the *differences in the API* before diving deeper.

2.3 Differences from Django admin

Django Candy is not a drop-in replacement for Django's default admin. This document lists the differences in the API.

2.3.1 `ModelAdmin` class

Candy provides its own *`ModelAdmin`* and `AdminSite` classes. You can't use Django's `ModelAdmin` with Candy.

Import `admin` from `django_candy`:

```
from django_candy import admin
```

2.3.2 List search

`search_fields` and `get_search_results` don't work in Candy.

Use *`get_filtered_queryset()`* to implement searching and filtering.

See *Usage docs on list search* for details.

2.3.3 List filters

Django admin's `list_filter` doesn't work. Instead, use `list_filters` (note the extra "**s**" at the end).

And Candy doesn't provide automatic filtering. You're required to filter the results yourself using *`ModelAdmin()`*.

See *Usage docs on list filters* for details.

2.4 Usage

Using Candy admin is similar to using Django's default admin.

2.4.1 Example models

Let's establish some models which will be used throughout this document for examples:

```
# models.py

from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=50)
    gender = models.CharField(max_length=10)
    email = models.EmailField()

class Post(models.Model):
    author = models.ForeignKey(Author, on_delete=models.CASCASE)
    title = models.CharField(max_length=150)
    content = models.TextField()
```

2.4.2 Registering models

Registering models is similar to Django, except you use *django_candy.admin*:

```
# admin.py

from django_candy import admin
from myapp.models import Author, Post

admin.site.register(Author)

# Or for more control, use ModelAdmin class

class PostAdmin(admin.ModelAdmin):
    list_display = ['title', 'author']

admin.site.register(Post, PostAdmin)
```

2.4.3 Customising list table columns

You can use `ModelAdmin.list_display` to specify which fields appear in list table:

```
from django_candy import admin

class AuthorAdmin(admin.ModelAdmin):
    list_display = ['name', 'email']

admin.site.register(Author, AuthorAdmin)
```

2.4.4 List search

Currently, Candy doesn't provide automatic search like Django's default admin does. Therefore, `search_fields` and `get_search_results` don't work.

To implement search for your models, Candy provides a *get_filtered_queryset()* method. It receives a `request`, `queryset` and `query_params` arguments. The `query_params` argument is a dict containing query parameters sent with the request. The name for the search query parameter is `q` which you can use for filtering the queryset.

This method returns a queryset.

```
class AuthorAdmin(admin.ModelAdmin):
    def get_filtered_queryset(self, request, queryset, query_params):
        search_term = query_params.get('q')

        if search_term:
            queryset = queryset.filter(name__istartswith=search_term)

        return queryset
```

2.4.5 List filters

Django's `list_filter` option doesn't work. Instead, use `list_filters` (note the extra "**s**" at the end).

Candy doesn't provide automatic filtering either.

`list_filters` option should be a list which contains dicts of all the filters and options.

Then, you can use the *`get_filtered_queryset()`* method to filter the results.

```
class AuthorAdmin(models.Model):
    list_filters = [
        {
            'label': 'Gender', 'name': 'gender', 'type': 'checkbox',
            'options': [
                {'label': 'Any', 'value': '', 'default': True},
                {'label': 'Male', 'value': 'male'},
                {'label': 'Female', 'value': 'female'},
            ]
        },
        {
            'label': 'Sort by', 'name': 'sort_by', 'type': 'radio',
            'options': [
                {'label': 'Name (A-Z)', 'value': 'name'},
                {'label': 'Name (Z-A)', 'value': '-name'},
            ]
        }
    ]

    def get_filtered_queryset(self, request, queryset, query_params):
        gender = query_params.get('gender')
        sort_by = query_params.get('sort_by')

        if gender:
            queryset = queryset.filter(gender__in=gender)

        if sort_by:
            queryset = queryset.order_by(sort_by)

        return queryset
```

2.5 Customising the admin site

2.5.1 Changing site name

Use the setting `CANDY_SITE_NAME` to set a name for the admin site.

```
CANDY_SITE_NAME = 'My Admin Site'
```

2.5.2 Changing logos

Logos are just static images and, so, can be overrided just like you override static files and templates in Django.

First create a folder called `django_candy` inside your Project's main `static` folder such that your project structure looks like this:

```
project/
    static/
        django_candy/
```

Now you can put the images in this `django_candy` directory by these names:

- `logo-icon.png` - Logo icon.
- `apple-touch-icon.png` - Image used for bookmarks on apple devices.
- `logo-nav.png` - Logo displayed on top navbar.
- `logo-nav-mobile.png` - Logo displayed on top navbar on mobile.
- `logo-splash.png` - Logo displayed on initial loading and login screen.
- `error-splash.png` - Image displayed on error pages.

2.6 API reference

This page provides a reference to the API. See the *Usage* page for a detailed guide on using the API.

class ModelAdmin

Replacement for DJango's `ModelAdmin`.

list_display

Used for specifying which fields appear in the list page table. It works similar to Django admin's.

It can accept field names and method names.

list_filters

Used for configuring the filters for the list page. It's **not similar** to Django admin.

See *Usage docs* for details.

get_filtered_queryset(*request*, *queryset*, *query_params*)

This method can be used for *implementing search* and *filters*.

Parameters

- **request** – The current request instance.
- **queryset** – The queryset for the list page.
- **query_params** – The query parameters sent with the request.

2.7 Release notes

2.7.1 Django Candy 0.7.2 release notes

Mar 08, 2021

Candy is still a pre-alpha project.

This version adds some *new features*.

What's new in v0.7.2

Login and logout

Admin now supports logging in and out.

List page search and filters

Support for searching a filtering on the list page.

Frontend build v0.7.1

Frontend build is updated to v0.7.1.

Note: Django Candy's frontend site is developed separately from the backend app. So the versions of frontend and backend may not always match.

2.7.2 Django Candy 0.7.1 release notes

Mar 04, 2021

Bug fixes

- Included django-rest-framework dependency in install requirements.

PYTHON MODULE INDEX

d

INDEX