django-crudbuilder Documentation

Release 0.1.5

Asif

Feb 11, 2018

Contents

CHAPTER 1

Contents:

1.1 Overview

Django-Crudbuilder allows you to generate class based views for CRUD (Create, Read, Update and Delete) for specific model

1.1.1 Features

- Generates class based views for CRUD
- Uses **django-tables2** to display objects in ListView
- Define multiple crud builders for same model with separate URL
- Allows custom forms/tables as additional arguments
- Context provides additional template variables **APP_LABEL** and **MODEL** for all CRUD templates
- Enable/disable login required option for CRUD views
- Enable/disable permission required option for CRUD views
- All the generated views/tables/forms/url are extendable.
- post_create and post_update signals to handle specific actions in Create and Update views
- Add your own templates for List/Create/Detail/Update/Delete views
- Separate CREATE and UPDATE forms
- Define your own custom queryset for list view
- Inline Formset support for parent child models
- Default Bootstrap3 CSS
- All the generated views are extendable.

1.1.2 Requirements and Compatibility

The 0.0.x series currently supports Django >= 1.8.x and corresponding versions of Python also supported by Django (including Python 3). Development of django-crudbuilder follows Django's Supported Version Policy and testing for older versions of Django/Python will be removed as time marches on.

1.2 Installation and Usage

Install django-crudbuilder:

```
pip install django-crudbuilder
```

Add django-crudbuilder to your `INSTALLED_APPS`:

```
# settings.py
INSTALLED_APPS = (
    ...
    'django_tables2',
    'crudbuilder'
)
```

View the *additional settings section* for a list of the django-crudbuilder settings that are available.

For a working implementation, you can view the example project on Github.

1.2.1 Generating CRUD class based views

The main business-logic for generating class based views for CRUD is done in `crudbuilder.views.ViewBuilder()`. You can use this class method directly in your own Views or you can use one of the Views packaged with this app.

We're going to run through creating a tutorial app. Let's start with a simple model:

```
# tutorial/models.py
class Person(models.Model):
    name = models.CharField(blank=True, max_length=100)
    email = models.EmailField()
    created_at = models.DateTimeField(auto_now=True)
    created_by = models.ForeignKey(User, blank=True, null=True)
```

Then create the CRUD class for `Person` model:

```
# tutorial/crud.py
from crudbuilder.abstract import BaseCrudBuilder

class PersonCrud(BaseCrudBuilder):
    model = Person
    search_fields = ['name']
    tables2_fields = ('name', 'email')
    tables2_css_class = "table table-bordered table-condensed"
    tables2_pagination = 20  # default is 10
    modelform_excludes = ['created_by', 'updated_by']
    login_required=True
    permission_required=True
```

```
    @classmethod
    def custom_queryset(cls, request, **kwargs):
        """Define your own custom queryset for list view"""
        qset = cls.model.objects.filter(created_by=request.user)
        return qset

    @classmethod
    def custom_context(cls, request, context, **kwargs):
        """Define your own custom context for list view"""
        context['custom_data'] = "Some custom data"
        return context

    # permissions = {
    #     'list': 'example.person_list',
    #     'create': 'example.person_create'
    # }
    # createupdate_forms = {
    #     'create': PersonCreateForm,
    #     'update': PersonUpdateForm
    # }
```

Finally implement the urls for the CRUD:

```
# tutorail/urls.py
urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^crud/',  include('crudbuilder.urls')),
    ]
```

The above will generate following URL's:

```
http://127.0.0.1:8000/crud/yourappname/yourmodelname
http://127.0.0.1:8000/crud/yourappname/yourmodelname/create/
http://127.0.0.1:8000/crud/yourappname/yourmodelname/<pk>/detail/
http://127.0.0.1:8000/crud/yourappname/yourmodelname/<pk>/update/
http://127.0.0.1:8000/crud/yourappname/yourmodelname/<pk>/delete/
```

View all your registered CRUDS:

```
http://127.0.0.1:8000/crud/
```

1.2.2 CRUD class Attributes

Usage of all below attributes you can view in CRUD class of example project on Github.

- **model** – Actual model name
- **search_fields** – Search fields for list view
- **custom_postfix_url** – Your own custom url postfix, if this value set then the resulted URL will become */crud/appname/<custom_postfix_url>*.
- **tables2_fields** – Fields which will be used in django-tables2 fields attribute (in list view)
- **tables2_css_class** – CSS class for list view table (for django-tables2)
- **tables2_pagination** – By default crudbuilder will set pagination to 10, you can oveeride this value by setting this attribute

- **modelform_excludes** – Exclude fields for model form
- **detailview_excludes** – Exclude fields in Deatail view
- **custom_modelform** – Your custom model form
- **custom_table2** – Your custom Tables2 class
- **custom_templates** – Your own custom templates. For more details on custom templates, you can check *custom templates*
- **login_required** – Enable login required for specific model CRUD (by default False)
- **permission_required** – Enable permission required for specific model CRUD (by default False)
- **permissions** – By default crudbuilder will generate crud permissions, if you want to define your own permissions then add permissions dictionary on the CRUD class. For more details on permission, you can check *custom permission*
- **createupdate_forms** – Define separate CREATE and UPDATE forms
- **custom_queryset** – Define your own custom queryset for list view
- **custom_context** – Define your own custom context for list view
- **inlineformset** – Define your Inline Formset for parent child relation, you can check *inline-formset-parent-child-relation* for more detail.

1.3 Additional Settings

There are a few additional settings you can use to add additional functionali for login and permission required and are set in your settings.py file.

1.3.1 LOGIN_REQUIRED_FOR_CRUD

- If `LOGIN_REQUIRED_FOR_CRUD` has been marked as `True` in settings.py, then login_required decorator will be enabled on all CRUD views globally.
- If you want to enable login required only for specific model crud, then you need to add following to crud class

```
# myapp/crud.py
login_required = True
```

- By default `LOGIN_REQUIRED_FOR_CRUD` is `False`, which means any user can view all the CRUD views.

1.3.2 PERMISSION_REQUIRED_FOR_CRUD

- If `PERMISSION_REQUIRED_FOR_CRUD` has been marked as `True` in settings.py, then permission_required decorator will be enabled on all CRUD views globally.
- If you want to enable permission required only for specific model crud, then you need to add following to crud class

```
# myapp/crud.py
permission_required = True
```

- By default `PERMISSION_REQUIRED_FOR_CRUD` is `False`, which means any user can view all the CRUD views.

By enabling either of above flag, by default crudbuilder checks for following permissions:

- For ListView : <your app_name>.<your model>_list
- For CreateView : <your app_name>.<your model>_create
- For DetailView : <your app_name>.<your model>_detail
- For UpdateView : <your app_name>.<your model>_update
- For DeleteView : <your app_name>.<your model>_delete

If you want to add your own permissions, then define your own permission required dictionary exlicitly in CRUD class.:

```
permissions = {
'list'  : 'example.permission1',
'create': 'example.permission2'
'detail': 'example.permission3',
'update': 'example.permission4',
'delete': 'example.permission5',
}
```

1.4 Forms

If NO custom forms defined in CRUD class, then by default crudbuilder will generate modelform from Django modelform factory.

1.4.1 Custom Modelform in CRUD class

You can define your own custom modelform in yourapp/forms.py and the same will be used for CRUD class. As shown below:

```
# yourapp/forms.py
class PersonEmploymentForm(forms.ModelForm):
        class Meta:
                model = PersonEmployment
                fields = '__all__'
                # exclude = ('person',)

        def __init__(self, *args, **kwargs):
                self.request = kwargs.pop('request', None)
                super(PersonEmploymentForm, self).__init__(*args, **kwargs)

# yourapp/crud.py
class PersonEmploymentCrud(BaseCrudBuilder):
        model = PersonEmployment
        custom_modelform = PersonEmploymentForm
```

Note: If you want to plug custom modelform, then make sure to override form's `__init__` to get *request* param as mentioned above.

1.4.2 Separate CREATE and UPDATE forms

You can also define separate forms for CreateView and UpdateView.:

```
# yourapp/forms.py
class PersonEmployementCreateForm(forms.ModelForm):
        class Meta:
                model = PersonEmployment
                exclude = ('person', 'medical_allowance')

        def __init__(self, *args, **kwargs):
                self.request = kwargs.pop('request', None)
                super(PersonEmployementCreateForm, self).__init__(*args, **kwargs)

class PersonEmployementUpdateForm(forms.ModelForm):
        class Meta:
                model = PersonEmployment
                exclude = ('salary', 'year')

        def __init__(self, *args, **kwargs):
                self.request = kwargs.pop('request', None)
                super(PersonEmployementUpdateForm, self).__init__(*args, **kwargs)

# yourapp/crud.py
class PersonEmploymentCrud(BaseCrudBuilder):
        model = PersonEmployment
        createupdate_forms = {
                'create': PersonEmployementCreateForm,
                'update': PersonEmployementUpdateForm
        }
```

You can check forms.py of example project on Github.

Note: If you want to plug custom modelform, then make sure to override form's `__init__` to get *request* param as mentioned above.

1.4.3 Inline Formset (Parent child relation)

The latest version of django-crudbuilder supports inline formset. You can define your own InlineFormset and give it to CRUD class.:

```
# yourapp/crud.py
from crudbuilder.formset import BaseInlineFormset

class PersonEmploymentInlineFormset(BaseInlineFormset):
        inline_model = PersonEmployment
        parent_model = Person
        exclude = ['created_by', 'updated_by']
        #formset_class = YourBaseInlineFormset
        #child_form = ChildModelForm

class PersonCrud(BaseCrudBuilder):
        model = Person
        search_fields = ['name']
```

```
        tables2_fields = ('name', 'email')
        inlineformset = PersonEmploymentInlineFormset
```

You can check crud.py of example project on Github.

You can set the following arguments for InlineFormset class.:

```
extra = 3               --> default number of forms of Child model
can_delete = True       --> Delete check box for child instance
formset_class = None    --> Django BaseInlineFormset class to override the methods of␣
↪InlineFormset (For example, if you want to override clean())
child_form              --> Modelform for Child model
inline_model = None     --> Child Model
parent_model = None     --> Parent Model
exclude = []            --> Exclude fields for Inline Formset
fields = None           --> Fields to disaply in inline formset
fk_name = None          --> More than one foreign key for the same parent model, then␣
↪specify this
```

1.5 Templates

By default django-crudbuilder uses Bootstrap3 style for its CRUD templates. You can view these templates in template folder of crudbuilder on Github.

1.5.1 Use your own HTML templates for crudbuilder

You can use your own templates for the crudbuilder in following two ways:

1.5.2 5 common templates for all models CRUD

You can create your own 5 common HTML templates for CRUD in templates/crudbuilder, then crudbuilder will use your defined templates.

Model

For single object crud.:

```
templates/crudbuilder/instance
        list.html
        create.html
        update.html
        delete.html
        detail.html
```

Inline Formset

For inline formset.:

```
templates/crudbuilder/inline
    list.html
    create.html
    update.html
    delete.html
    detail.html
```

Custom templates for specific model:

If you want to create custom templates for specific model, then update the CRUD class with custom template path as shown below.:

```
class PersonCrud(BaseCrudBuilder):
    model = Person
    search_fields = ['name']
    tables2_fields = ('name', 'email')
    tables2_css_class = "table table-bordered table-condensed"
    tables2_pagination = 20  # default is 10
    modelform_excludes = ['created_by', 'updated_by']

    custom_templates = {
        'list': 'yourtemplates/your_list_template.html',
        'create': 'yourtemplates/your_create_template.html',
        'detail': 'yourtemplates/your_detail_template.html',
        'update': 'yourtemplates/your_update_template.html',
        'delete': 'yourtemplates/your_delete_template.html'
        }
```

1.5.3 Enable search in ListView template

If you are writing your own custom templates, then please add the following to your list view template to enable the search.:

```
<form action="." method="GET">
    <input type="text" name='search'>
    <button type="submit" >Search</button>
</form>
```

1.6 Signals

django-crudbuilder comes with built-in signals to perform/execute specific actions after post create and post update views. You can able to access `request` and `instance` in your own handlers.

Usage of these signals you can view in handlers of example project on Github.

Following are the list of supported signals.:

```
post_create_signal         --> runs after modelform.save() in CreateView
post_update_signal         --> runs after modelform.save() in UpdateView
post_inline_create_signal  --> runs after inlineformset.save() in CreateView
post_inline_update_signal  --> runs after inlineformset.save() in UpdateView
```

1.6.1 post_create_signal

This signal will loaded/executed soon after the object gets created for specific model. In otherwords this will fires right after the `modelform.save()` method gets called during the CreateView.:

```
# tutorial/handlers.py
from django.dispatch import receiver
from crudbuilder.signals import post_create_signal

from example.models import Person

@receiver(post_create_signal, sender=Person)
def post_create_signal_handler(sender, **kwargs):
    # print "execute after create action"
    request = kwargs['request']
    instance = kwargs['instance']

    instance.created_by = request.user
    instance.save()
```

1.6.2 post_update_signal

This signal will loaded/executed soon after the object gets updated for specific model. In otherwords this will fires right after the `modelform.save()` method gets called during the UpdateView.:

```
# tutorial/handlers.py
from django.dispatch import receiver
from crudbuilder.signals import post_update_signal

from example.models import Person

@receiver(post_update_signal, sender=Person)
def post_update_signal_handler(sender, **kwargs):
    # print "execute after update action"
    request = kwargs['request']
    instance = kwargs['instance']

    instance.updated_by = request.user
    instance.save()
```

1.6.3 post_inline_create_signal

This signal will get executed soon after the inline formset gets saved which means this get fires right after the `inlineformset.save()` method gets called in CreateView.:

```
# tutorial/handlers.py
from django.dispatch import receiver
from crudbuilder.signals import post_inline_create_signal

@receiver(post_inline_create_signal, sender=Person)
def post_inline_create_handler(sender, **kwargs):
        request = kwargs['request']
        parent = kwargs['parent']
        children = kwargs['children']
```

```
        parent.created_by = request.user
        parent.save()

        for child in children:
                child.created_by = request.user
                child.save()
```

1.6.4 post_inline_update_signal

This signal will get executed soon after the inline formset gets updated which means this get fires right after the `inlineformset.save()` method gets called in UpdateView.:

```
# tutorial/handlers.py
from django.dispatch import receiver
from crudbuilder.signals import post_inline_update_signal

@receiver(post_inline_update_signal, sender=Person)
def post_inline_update_handler(sender, **kwargs):
        request = kwargs['request']
        parent = kwargs['parent']
        children = kwargs['children']

        parent.updated_by = request.user
        parent.save()

        for child in children:
                child.updated_by = request.user
                child.save()
```

CHAPTER 2

Issues:

Use the GitHub issue tracker for django-crudbuilder to submit bugs, issues, and feature requests.