CRUDLFA+ Documentation

Release 0.0

James Pic & Contributors

May 23, 2018

Contents:

CRUDLFA+ stands for Create Read Update Delete List Form Autocomplete and more.

This plugin for Django makes a rich user interface from Django models.

CHAPTER 1

Install CRUDLFA+ module

This section concerns This package can be installed from PyPi by running:

1.1 Installing from PyPi

If you are just getting started with CRUDLFA+, it is recommended that you start by installing the latest version from the Python Package Index (PyPi). To install CRUDLFA+ from PyPi using pip run the following command in your terminal.

```
pip install crudlfap
```

If you are not in a **virtualenv_**, the above will fail if not executed as root, in this case use `install --user`:

```
pip install --user crudlfap
```

1.2 With development packages

If you intend to run the `crudlfap dev` command, then you should have the development dependencies by adding `[dev]`:

```
pip install (--user) crudlfap[dev]
```

Then, you should see the example project running on port 8000 with command:

```
crudlfap dev
```

1.3 Installing from GitHub

You can install the latest current trunk of crudlfap directly from GitHub using pip.

```
pip install --user -e git+git://github.com/yourlabs/crudlfap.git@master
↪#egg=crudlfap[dev]
```

Warning: `[dev]`, `--user`, `@master` are all optionnal above.

1.4 Installing from source

1. Download a copy of the code from GitHub. You may need to install git.

```
git clone https://github.com/yourlabs/crudlfap.git
```

2. Install the code you have just downloaded using pip, assuming your current working directory has not changed since the previous command it could be:

   ```
   pip install -e ./crudlfap[dev]
   ```

Move on to the *CRUDLFA+ Tutorial*.

CHAPTER 2

CRUDLFA+ Tutorial

2.1 About document

This document attempts to teach the patterns you can use, and at the same time go through every feature. The document strives to teach CRUDLFA+ as efficiently as possible. If it becomes too long, we will see how we refactor the document, until then, it serves as main documentation. Please contribute any modification you feel this document needs to fit its purpose.

2.2 About module

CRUDLFA+ strives to provide a modern UI for Django generic views out of the box, but all defaults should also be overiddable as conveniently as possible. It turns out that Django performs extremely well already, but by pushing Django's philosophy such as DRY as far as possible, even in the client side code world.

2.3 Enable in your project

We're going to setup `TEMPLATES` and `INSTALLED_APPS` before we begin.

Note: We will review the minimal settings in this tutorial, but you can consult the default settings available for your crudlfap version in the *`settings`* module.

2.3.1 TEMPLATES

CRUDLFA+ uses Jinja2 templates with a quite extended configuration. Options to enable them are using any of these in your settings:

- easiest: *`crudlfap.settings.TEMPLATES`*

- intermediate: *crudlfap.settings.CRUDLFAP_TEMPLATE_BACKEND*
- custom: *crudlfap.settings.DEFAULT_TEMPLATE_BACKEND*

2.3.2 INSTALLED_APPS

CRUDLFA+ leverages apps from the Django ecosystem. Use *crudlfap.settings.CRUDLFAP_TEMPLATE_BACKEND*. To help make this a pleasant experience, CRUDLFAP+ splits the INSTALLED_APPS setting into multiple settings you can import and mix together:

- everything: *crudlfap.settings.INSTALLED_APPS*,
- crudlfap only: *crudlfap.settings.CRUDLFAP_APPS*,
- django apps: *crudlfap.settings.DJANGO_APPS*,

2.4 Define a Router

2.4.1 Register a CRUD with default views using Router.register()

Just add a `crudlfap.py` file in one of your installed apps, and the `DefaultConfig` will autodiscover them, this example shows how to enable the default CRUD for a custom model:

```
from crudlfap import crudlfap

from .models import Artist


crudlfap.Router(
    Artist,
    fields='__all__',
    # Optionnal hack to allow unauthenticated access:
    allowed=lambda view: True
).register()
```

In this case, the *Router* will get the views it should serve from the *CRUDLFAP_VIEWS* setting.

2.4.2 Custom view parameters with View.clone()

If you want to specify views in the router:

```
.. literalinclude:: ../src/crudlfap_example/song/crudlfap.py
```

Using the `clone()` classmethod will define a subclass on the fly with the given attributes.

2.5 URLs

The easiest configuration is to generate patterns from the default registry:

```
from crudlfap import crudlfap

urlpatterns = [
    crudlfap.site.urlpattern
]
```

Or, to sit in `/admin`:

```
urlpatterns = [
    crudlfap.site.get_urlpattern('admin'),
    # your patterns ..
]
```

CHAPTER 3

View URL autogeneration mechanisms: RoutableViewMixin

One of the key architectural concepts of CRUDLFA+ is the ability for views to generate their own URLs. This chapter reviews the different mechanisms in place and how they are overridable.

Code which makes a view encapsulate what it takes to make it auto generate urls is located in the `Route`, which we'll describe intensively here.

All black magic for views are defined in the crudlfap.route module.

CRUDLFA+ introduces a new design pattern for views that came out during refactoring sessions from a corporate project, and re-written for Django 2.0 from scratch. L

CHAPTER 4

URLPatterns autogeneration mechanisms: Router

One of the key architectural concepts of CRUDLFA+ is the ability to tie a group of view with a model class to autogenerate urlpatterns. This chapter reviews the different mechanisms in place and how they are overridable.

Source is located in the `Router`, which we'll describe here.

CRUDLFA+ router for Django 2.0.

Note that you can also use non-database backed models, by inheriting from models.Model and setting their Meta.managed attribute to False. Then, you can use CRUDLFA+ views and routers.

class `crudlfap.router.`**`Router`**(*model=None*, *registry=None*, *views=None*, ***attributes*)

Base router for CRUDLFA+ Route.

`model`

Optional model class for this Router and all its views.

`allowed`(*view*)

Return True to allowed a access to a view.

Called by the default view.allowed() implementation.

If you override the view.allowed() method, then it's up to you to decide if you want to call this method or not.

Returns True if user.is_staff by default.

`generate_views`(**views*)

Generate views for this router, core of the automation in CRUDLFA+.

This method considers each view in given args or self.views and returns a list of usable views.

Each arg may be a view class or a dict of attributes with a *_cls* key for the actual view class.

It will copy the view class and bind the router on it in the list this returns.

For example, this would cause two view classes to be returned, if self.model is `Artist`, then `CreateView` will be used as parent to create `ArtistCreateView` and `DetailView` will be used to create `ArtistDetailView`, also setting the attribute `extra_stuff='bar'`:

```
Router(Artist).generate_views([
    CreateView,
    dict(_cls=DetailView, extra_stuff='bar'),
    ListView.factory(paginate_by=12),
])
```

get_app_name()
Generate app name for this Router views.

get_fields_for_user(*user*, *perms*, *obj=None*)
Return the list of fields for a user.

get_menu(*name*, *request*, ***kwargs*)
Return allowed view objects which have `name` in their `menus`.

For each view class in self.views which have `name` in their `menus` attribute, instanciate the view class with `request` and kwargs, call `allowed()` on it.

Return the list of view instances for which `allowed()` has passed.

get_namespace()
Generate namespace for this Router views.

get_objects_for_user(*user*, *perms*)
Return the list of objects for a given set of perms.

get_urlfield()
Return Field name of model for reversing url.

This will return model ‘ slug ‘ field if available or ‘ pk ‘ field.

See `guess_urlfield()` for detail.

get_urlpath()
Return Model name for urlpath.

get_urlpatterns()
Generate URL patterns for this Router views.

register()
Register to self.registry.

Also, adds the get_absolute_url() method to the model class if it has None, to return the reversed url for this instance to the view of this Router with the `detail` slug.

Set get_absolute_url in your model class to disable this feature. Until then, welcome in 2018.

Also, register this router as default router for its model class in the RouterRegistry.

class crudlfap.router.**Views**

CHAPTER 5

Settings

5.1 Project

A settings file to import boilerplate from.

`crudlfap.settings.`**`CRUDLFAP_VIEWS`**

List of default views to provide to Routers that were not spawned with any view.

`crudlfap.settings.`**`INSTALLED_APPS`**

That list contains both *`CRUDLFAP_APPS`* and *`DJANGO_APPS`* and you can use them as such on a new project:

```
from crudlfap.settings import INSTALLED_APPS

INSTALLED_APPS = ['yourapp'] + INSTALLED_APPS
```

`crudlfap.settings.`**`CRUDLFAP_APPS`**

List of apps CRUDLFA+ depends on, you can use it as such:

```
from crudlfap.settings import CRUDLFAP_APPS

INSTALLED_APPS = [
    'yourapp',
    'django.contrib.staticfiles',
    # etc
] + CRUDLFAP_APPS
```

`crudlfap.settings.`**`DJANGO_APPS`**

This list contains all contrib apps from the Django project that CRUDLFA+ should depend on. You can use it as such:

```
from crudlfap.settings import CRUDLFAP_APPS, DJANGO_APPS

INSTALLED_APPS = ['yourapp'] + CRUDLFAP_APPS + DJANGO_APPS
```

`crudlfap.settings.`**`TEMPLATES`**

This list contains both *DEFAULT_TEMPLATE_BACKEND* and *CRUDLFAP_TEMPLATE_BACKEND* and works out of the box on an empty project. You can add it to your settings file by just importing it:

```
from crudlfap.settings import TEMPLATES
```

`crudlfap.settings.`**`CRUDLFAP_TEMPLATE_BACKEND`**

Configuration for Jinja2 and environment expected by CRUDLFA+ default templates. Add it to your own TEMPLATES setting using import:

```
from crudlfap.settings import CRUDLFAP_TEMPLATE_BACKEND

TEMPLATES = [
    # YOUR_BACKEND
    CRUDLFAP_TEMPLATE_BACKEND,
]
```

`crudlfap.settings.`**`DEFAULT_TEMPLATE_BACKEND`**

Configuration for Django template backend with all builtin context processors. You can use it to define only your third backend as such:

```
from crudlfap.settings import (
    CRUDLFAP_TEMPLATE_BACKEND,
    DEFAULT_TEMPLATE_BACKEND,
)

TEMPLATES = [
   # YOUR_BACKEND
   CRUDLFAP_TEMPLATE_BACKEND,
   DEFAULT_TEMPLATE_BACKEND,
]
```

`crudlfap.settings.`**`DEBUG`**

Evaluate `DEBUG` env var as boolean, False by default.

`crudlfap.settings.`**`SECRET_KEY`**

Get `SECRET_KEY` env var, or be `'notsecret'` by default.

Danger: Raises an Exception if it finds both SECRET_KEY=notsecret and DEBUG=False.

`crudlfap.settings.`**`ALLOWED_HOSTS`**

Split `ALLOWED_HOSTS` env var with commas, or be `['*']` by default.

Danger: Raises an Exception if it finds both ALLOWED_HOSTS to be `'*'` and DEBUG=False.

`crudlfap.settings.`**`MIDDLEWARE`**

A default MIDDLEWARE configuration you can import.

`crudlfap.settings.`**`OPTIONAL_APPS`**

from crudlfap.settings import * # [. . .] your settings install_optional(OPTIONAL_APPS, INSTALLED_APPS) install_optional(OPTIONAL_MIDDLEWARE, MIDDLEWARE)

CHAPTER 6

Views

Source is located in the `generic`, which we'll describe here.

Crudlfa+ generic views and mixins.

Crudlfa+ takes views further than Django and are expected to:

- generate their URL definitions and reversions,
- check if a user has permission for an object,
- declare the names of the navigation menus they belong to.

class `crudlfap.views.generic.`**`CreateView`**`(`***kwargs*`)`
View to create a model object.

class `crudlfap.views.generic.`**`DefaultTemplateMixin`**
Override for get_template_names to append default_template_name.

This allows to configure "last resort" templates for each class, and thus to provide a working CRUD out of the box.

`get_template_names``()`
Give a chance to default_template_name.

`get_title_heading``()`
Return text for page heading.

`get_title_html``()`
Return text for HTML title tag.

`get_title_link``()`
Return title attribute for links to this view.

`get_title_menu``()`
Return title for menu links to this view.

class `crudlfap.views.generic.`**`DeleteAction`**
View to delete a model object.

class `crudlfap.views.generic.`**`DeleteView`**`(`***kwargs*`)`

class `crudlfap.views.generic.`**`DetailView`**(***kwargs*)
Templated model object detail view which takes a field option.

`get_context_data`(**a*, ***k*)
Insert the single object into the context dict.

classmethod get_urlpath()
Identify the object by slug or pk in the pattern.

class `crudlfap.views.generic.`**`FormView`**(***kwargs*)
Base FormView class.

class `crudlfap.views.generic.`**`FormViewMixin`**
Mixin for views which have a Form.

class `crudlfap.views.generic.`**`HistoryView`**(***kwargs*)

class `crudlfap.views.generic.`**`ListDeleteView`**(***kwargs*)

class `crudlfap.views.generic.`**`ModelFormView`**(***kwargs*)

class `crudlfap.views.generic.`**`ModelFormViewMixin`**
ModelForm ViewMixin using readable

`message_html`(*message*)
Add the detail url for form.instance, if possible.

class `crudlfap.views.generic.`**`ModelViewMixin`**
Mixin for views using a Model class but no instance.

`get_queryset`()
Return router.get_queryset() by default, otherwise super().

class `crudlfap.views.generic.`**`ObjectFormView`**(***kwargs*)
Custom form view on an object.

class `crudlfap.views.generic.`**`ObjectFormViewMixin`**
Custom form view mixin on an object.

class `crudlfap.views.generic.`**`ObjectMixin`**
Make self.object call and cache self.get_object() automatically.

WHAT A RELIEF

However, if it has a router with the get_object() method, use it.

`get_object`()
Return router.get_object() by default, otherwise super().

`object`
Return the object, uses get_object() if necessary.

`object_get`()
Return the object, uses get_object() if necessary.

`object_set`(*value*)
Set self.object attribute.

class `crudlfap.views.generic.`**`ObjectView`**(***kwargs*)

class `crudlfap.views.generic.`**`ObjectViewMixin`**
Mixin for views using a Model instance.

`get_slug_field`()
Replace Django's get_slug_field with get_url_field.

get_urlargs()
Return list with object's urlfield attribute.

classmethod get_urlpath()
Identify the object by slug or pk in the pattern.

slug_url_kwarg
Replace Django's slug_url_kwarg with get_url_field.

classmethod to_url_args(**args*)
Return first arg's url_field attribute.

class crudlfap.views.generic.**TemplateView**(***kwargs*)
TemplateView for CRUDLFA+.

class crudlfap.views.generic.**UpdateView**(***kwargs*)
Model update view.

class crudlfap.views.generic.**View**(***kwargs*)
Base view for CRUDLFA+.

class crudlfap.views.generic.**ViewMixin**
Base View mixin for CRUDLFA+.

If you have any question about style then find your answers in DefaultTemplateMixin, otherwise in Routable-ViewMixin.

CHAPTER 7

crudlfap_auth: crudlfap module for django.contrib.auth

7.1 Auth Views

Source is located in the `views`, which we'll describe here.

Crudlfa+ PasswordView, Become and BecomeUser views.

Crudlfa+ takes views further than Django and are expected to:

- generate their URL definitions and reversions,
- check if a user has permission for an object,
- declare the names of the navigation menus they belong to.

class `crudlfap_auth.views.`**`Become`**(***kwargs*)

class `crudlfap_auth.views.`**`BecomeUser`**(***kwargs*)

> **`get_object`**(*queryset=None*)
> Return router.get_object() by default, otherwise super().
>
> **`get_title_menu`**()
> Return title for menu links to this view.

class `crudlfap_auth.views.`**`PasswordView`**(***kwargs*)

> **`get_form_class`**()
> Return the form class to use in this view.
>
> **`get_form_kwargs`**()
> Return the keyword arguments for instantiating the form.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index