Supra Documentation

Release 1.0

Mario Barrios

October 28, 2016

Contents

1 Getting Started **1**
- 1.1 Installation . . . 1

2 Documentation **3**
- 2.1 SupraListView . . . 3
- 2.2 SupraFormView . . . 5
- 2.3 SupraInlineFormView . . . 5
- 2.4 Body Request . . . 6
- 2.5 SupraSession . . . 7

3 Licence **9**

4 Help **11**

CHAPTER 1

Getting Started

1.1 Installation

1. You can get stable version of Supra by using pip or easy_install:

```
pip install supra
```

2. You will need to add the `'supra'` application to the `INSTALLED_APPS` setting of your Django project `settings.py` file.:

```
INSTALLED_APPS = (
    ...
    'supra',
)
```

CHAPTER 2

Documentation

2.1 SupraListView

It's a simple paginater JSON service. It shows a searchable list of registers paginated by an indicated number.

Fields

- *model:* Stipulate the model which will be shown, **it is mandatory**.
- *list_display:* Stipulate the field list to show of this model.
- *list_filter:* Stipulate the searchable field list for multiples params.
- *search_fields:* Stipulate the searchable field list form single param.

Example

`models.py` file:

```
from django.db import models

class MyModel(models.Model):
  field1 = models.CharField(max_length=45)
  field2 = models.CharField(max_length=45)
  field3 = models.CharField(max_length=45)
```

`view.py` file:

```
import supra
import models

class MyModelListView(supra.SupraListView):
  model = models.MyModel
  list_display = ['field1', 'field2', 'field3']
  search_fields = ['field1', 'field2']
  list_filter = ['field1', 'field2']
```

`urls.py` file:

```
from django.conf.urls import include, url
import views

urlpatterns = [
  url(r'mymodel/list/', views.MyModelListView.as_view(), name="mymodel_list"),
]
```

Query for multiple params.

```
/?field1=1
```

Query for single param.

```
/?search=1
```

Result.:

```
{"num_rows": 1, "object_list": [{"field1": "value1", "field2":"value2", "field3":"value3"}]}
```

For change the search param's name you can use search_key.

`view.py` file:

```
import supra
import models

class MyModelListView(supra.SupraListView):
  model = models.MyModel
  search_key = 'q'
  list_display = ['field1', 'field2', 'field3']
  search_fields = ['field1', 'field2']
  list_filter = ['field1', 'field2']
```

Query for single param.

```
/q=1
```

Also you can use *field__field* instead field name as *list_display* as for *search_fiels* and *list_filter*.

`view.py` file:

```
import supra
import models

class MyModelListView(supra.SupraListView):
  model = models.MyModel
  list_display = ['field1__subfield', 'field2',]
```

Result.

```
{"num_rows": 1, "object_list": [{"field1__subfield": "subvalue", "field2":"value2"}]}
```

If you don't want to show JSON keys like *field__subfield*, you can use **Renderer** sub class.

Renderer Sub class Renderer let you use friendly names for you JSON keys instead *field__subfield*.

`view.py` file:

```
class MyModelListView(supra.SupraListView):
  model = models.MyModel
  list_display = ['friendly', 'field2',]
  class Renderer:
    friendly = 'field1__subfield'
```

Result.

```
{"num_rows": 1, "object_list": [{"friendly": "subvalue", "field2":"value2"}]}
```

Pagination

You can paginate your service jus using the *paginate_by* attribute like this:

```
from supra import views as supra
import models
class MyModelListView(supra.SupraListView):
  model = models.MyModel
  list_display = ['friendly', 'field2',]
  paginate_by = 2
```

You can use the *page* GET parameter to select which page choose, the page range start with 1.

```
/?page=1
```

Result.

```
{"count": 5, "num_pages": 3, "object_list": [{"field2": "", "field3": "", "field1": "1"}, {"field2":
```

2.2 SupraFormView

It's a class based on the native django FormView class, but modified for use JSON as error list response instead of a HTML template.

Fields

- *model:* Espesify the model which will be created and/or edited, **it is mandatory**.
- *form_class:* Espesify the form class which will create and/or edit the model, **it is mandatory**.
- *template_name:* Espesify the name/path file for render the form template. it is not mandatory, if you do not espesify it, supra will use a generic default template.
- *inlines:* Espesify a **SupraInlineFormView** list for stack in this form.

Example

`view.py` file:

```
class MyModelFormView(supra.SupraFormView):
      model = models.MyModel
      form_class = forms.MyModelForm
      template_name = 'MyModelTemplate.html'
#end class
```

MyModelTemplate.html

```
<form action="" method="post">
    {% csrf_token %}
    {{form.as_p}}
    <input type="submit" value="Send message" />
</form>
```

On error will show a response like.

```
{"field1":["This field is required."]}
```

2.3 SupraInlineFormView

Fields

- *base_model:* Spesify the base model for attach the set of others models, **it is mandatory**.
- *model:* Spesify the model which will be attached on the base model, **it is mandatory**, also is mandatory that the inline model have a relation(ForeignKey, OneToOneField, ...) directly with the base model.
- *form_class:* Spesify the form class which will be used for create the formset.

Example

`models.py` file:

```
from django.db import models

class MyModel(models.Model):
  field1 = models.CharField(max_length=45)
  field2 = models.CharField(max_length=45)
  field3 = models.CharField(max_length=45)

class MyInlineModel(models.Model):
  mymodel = models.ForeignKey(MyModel)
  inlinefield = models.CharField(max_length=45)
```

`view.py` file:

```
class MyInlineModelFormView(supra.SupraInlineFormView):
      base_model = models.MyModel
      model = models.MyInlineModel
      form_class = forms.MyInlineModelForm

class MyModelFormView(supra.SupraFormView):
      model = models.MyModel
      form_class = forms.MyModelForm
      template_name = 'MyModelTemplate.html'
      inlines = [MyInlineModelFormView]
```

`MyModelTemplate.html`

```
<form action="" method="post">
{% csrf_token %}
    {{form.as_p}}
    {% for fo in inlines %}
      {{ fo.as_p }}
    {% endfor %}
    <input type="submit" value="Send message" />
</form>
```

On error will show a response like.

```
{"field1":["This field is required."], "inlines":[{"inlinefield": "This field is required."}]}
```

2.4 Body Request

The *body request* is not enable by default, but you can enable it using the *body* attribute in *SupraListView* class.

Example

`view.py` file:

```
class MyModelFormView(supra.SupraFormView):
  model = models.MyModel
  form_class = forms.MyModelForm
  template_name = 'MyModelTemplate.html'
  body = True
```

Also you can use the *SupraConf* class for configure for all like this.:

```
supra.SupraConf.body = True
```

2.5 SupraSession

Let us create a easy Django login.

Example

`view.py` file:

```
class MySupraSession(supra.SupraSession):
    pass
```

`urls.py` file:

```
urlpatterns = [
      url(r'session/', supra.SupraSession.as_view()),
]
```

Now you can login sending a POST request to the 'session/' url and logout senfing a DELETE request to the same url.

Also if want to validate one soesific user type you can use the model attribute for spesify wich user can be login with this url, liki this:

`view.py` file:

```
class MySupraSession(supra.SupraSession):
    model = models.MyUser
```

Then only the MyUser users can login with that url. That's all for now folks.

CHAPTER 3

Licence

The MIT License (MIT)

Copyright (c) 2016 ExilDev

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER 4

Help

If you need help, contact us! mariobarrpach@gmail.com