How to create and manipulate modal windows using django-fm

Summary

This tutorial on how to create modal windows in django applications is based on the repository on modal AJAX form to create, update and delete Django objects with ease, which can be found here.

Installation

1st Step: Run in the console the command ***pip install django-fm***

2nd Step: Go to the settings.py file of your project and add to the INSTALLED_APPS section the *crispy_forms* and *fm* apps as below.

```
INSTALLED_APPS = (
    ...
    'crispy_forms',
    'fm',
)
```

3rd Step: Also in settings.py set crispy template pack to bootstrap3:

```
CRISPY_TEMPLATE_PACK = 'bootstrap3'
```

4th Step: Include modal template file(for example you can call it **modal_base.html**) into your project template directory and initialize jQuery plugin. In other words, you have to create a template which will include all the appropriate libraries(bootstrap,jquery) in order to be able to create the AJAX forms. You can find the code for the **modal_base.html** template below. A common practice is to extend this code in order to take advantage of the features of modal-fm also to other templates you are going to create for your project.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <link rel="stylesheet"
href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css"/>
    <script type="text/javascript"
src="//ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js"></script>
    <script type="text/javascript"
src="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/js/bootstrap.min.js"></script
>
  </head>
  <body>
    {% block content %}{% endblock %}
    {% include "fm/modal.html" %}
    <script type="text/javascript">
      $(function() {
        $.fm({debug: true});
      });
    </script>
  </body>
</html>
```

Conclusion: There are 3 class-based views in django-fm to inherit from when you want AJAX forms:

- AjaxCreateView
- AjaxUpdateView
- AjaxDeleteView

You create urls for them as usual, in templates you just create links to create, update, delete resources with special class (fm-create, fm-update, fm-delete).

Performance (including code snippets)

In general, in our project we want to simulate the creation, updating and deleting of an instance of **User** class, which is defined in our models.py file. At this point we are going to explain the procedure of creating the forms based on a model with certain attributes. The AjaxCreateView, AjaxUpdateView and AjaxDeleteView procedures are similar.

models.py

In our case we create a class called **User** with attributes, the name of the user, the last name of the user and the email of the user. In your application you can add or subtract attributes according to your needs.

```
from django.db import models

class User(models.Model):
    '''This class simulates a user with a name,username and e-mail as
attributes'''
    name = models.CharField(max_length=200)
    lastname=models.CharField(max_length=200)
    email = models.EmailField(max_length=70,unique=True)

    def __str__(self):
        '''this function returns the name,lastname and email of the instance'''
        return self.name+" "+self.lastname+" "+self.email
```

forms.py

The form for the class **User** is defined here.

```
from .models import User
from django.forms import ModelForm

class UserForm(ModelForm):
        class Meta:
        model = User
        fields = ['name','lastname','email']
```

views.py

The views in this project are class-based. The joblist here is the name of the application. We are going to create a view to create,update and delete a new instance of our model.

```
from fm.views import AjaxCreateView,AjaxUpdateView,AjaxDeleteView
from django.urls import reverse_lazy
from joblist.models import User

### CRUD USERS ###

class UserCreateView(AjaxCreateView):
   model=User
   form_class = UserForm

class UserUpdateView(AjaxUpdateView):
        model = User
        form_class = UserForm
        success_url = reverse_lazy('users')

class UserDeleteView(AjaxDeleteView):
        model = User
        success_url = reverse_lazy('users')
```

urls.py

At this point we have to connect the class based views with the urls defined in the urls.py file.

```
from django.contrib import admin
from django.conf.urls import url,include
from joblist import views

urlpatterns = [

 url(r'^user/new/$',views.UserCreateView.as_view(),name="user_new"),
url(r'^user/edit/(?P<pk>\d+)/$', views.UserUpdateView.as_view(),
name='user_edit'),
url(r'^user/delete/(?P<pk>\d+)/$', views.UserDeleteView.as_view(),
name='user_delete'),

]
```

templates

In general, you don't have to define template for your form - just write a link to create new object with special attributes which tell django-fm how to behave.
So in your template write:

<a href="{% url user_new %}" class="fm-create" data-fm-head="Create" data-fm-callback="reload">Create new</a>

As you can see the user_new in the url template tag is the name reffering to UserCreateView view which we have defined in the urls.py file above. The same procedure is followed for the UserUpdateView and the UserDeleteView.

Look at fm-create special class - it's necessary. And that's all - now when user clicks on this link - modal AJAX window with form will be shown.
Every link can have some attributes which define modal window behaviour and callback after successfull object creation, update or deletion:

- data-fm-head - header of modal
- data-fm-callback - what to do after successfull modal submission - at moment the following values allowed: reload, redirect, replace, remove, prepend, append, redirect_from_response
- data-fm-target - value of action specific for each action type - for example this must be an URL when data-fm-callback is redirect

user_list.html

This template is the custom template of our application extending the **modal_base.html**. Here with the red colour we have the implementation of fm-create,fm-update and fm-delete.

{% extends "modal_base.html" %}

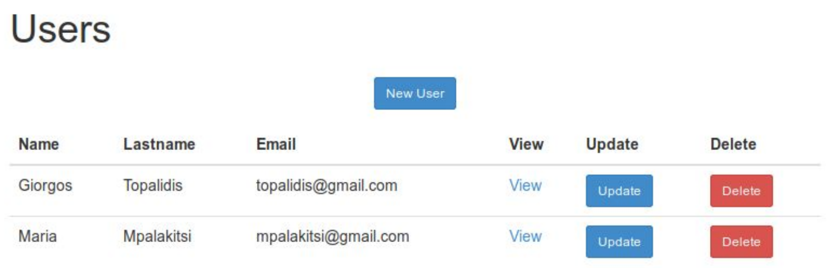
{% block content %}
<h1>Users</h1>

<div style="margin: 0 auto; text-align:center; padding:1em;">
<a href="{% url 'user_new' %}" class="fm-create" data-fm-head="New User" data-fm-callback="reload">

```
                <button class="btn btn-primary btn-sm" type="">New
User</button></a>
        </div>
<div class="table-responsive">
<table class="table">
<thead>
        <tr>
        <th>Name</th>
        <th>Lastname</th>
        <th>Email</th>
        <th>View</th>
        <th>Update</th>
        <th>Delete</th>
        </tr>
</thead>
<tbody>
        {% for user in users %}
        <tr>
        <td>{{ user.name }}</td>
        <td>{{ user.lastname }}</td>
        <td>{{ user.email }}</td>
        <td><a href="{% url 'user_view' user.id %}">View</a></td>

        <td><a href="{% url 'user_edit' user.id %}" class="fm-update"
data-fm-head="Edit User: {{ user }} ?" data-fm-callback="reload">
                <button class="btn btn-primary btn-sm"
type="">Update</button></a></td>
        <td><a href="{% url 'user_delete' user.id %}" class="fm-delete"
data-fm-head="Delete of User: {{ user }} ?" data-fm-callback="reload">
                <button class="btn btn-danger btn-sm"
type="">Delete</button></a></td>
        </tr>
        {% endfor %}
</tbody>
</table>
</div>
{% endblock %}
```

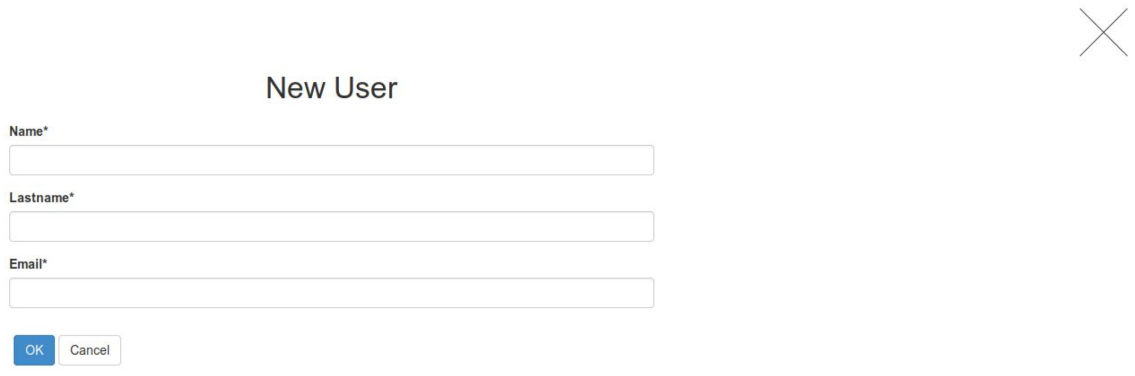
Screenshots



Users

Name	Lastname	Email	View	Update	Delete
Giorgos	Topalidis	topalidis@gmail.com	View	Update	Delete
Maria	Mpalakitsi	mpalakitsi@gmail.com	View	Update	Delete

picture 1: User list



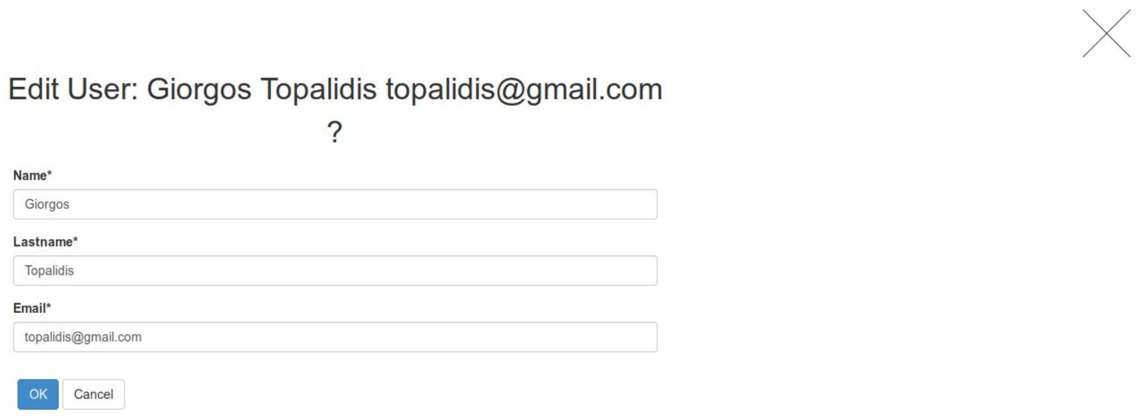
New User

Name*

Lastname*

Email*

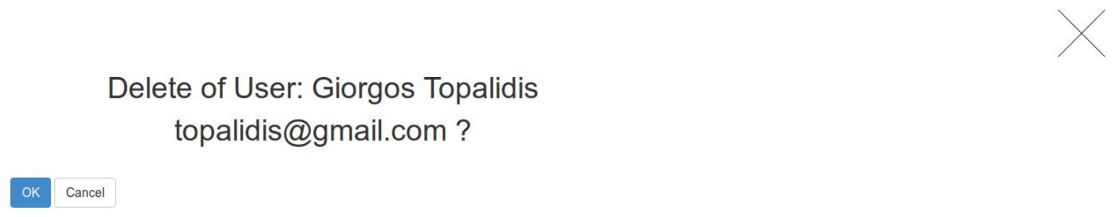
picture 2: New User window



Edit User: Giorgos Topalidis topalidis@gmail.com ?

Name*	Giorgos
Lastname*	Topalidis
Email*	topalidis@gmail.com

picture 3: Edit User window



Delete of User: Giorgos Topalidis topalidis@gmail.com ?

picture 4: Delete User window