django-ajax-form-mixin Documentation

Release 0.0.1

Jonas Geiregat

Sep 27, 2017

Contents

This is a fairly simple application for performing ajax validation of forms created using Django's forms system. Currently it only works with jQuery.

This is mostly a re-write of django-ajax-validation. Which replaces the view by a mixin and some improved jQuery callback handling.

Contents:

CHAPTER 1

Usage

To use django-ajax-formmixin, it requires you add a the AjaxFormMixin to your FormView, and some javascript to any page with the form.

For example, if you had the following form:

```
from django import forms

class ContactForm(forms.Form):
    name = forms.CharField(label='Your Name')
    email = forms.EmailField(label='Your Email')
    message = forms.CharField(label='Your Message', widget=forms.Textarea)
```

With the following View:

```
class ContactFormView(AjaxFormView):
    template_name = "contact-form.html"
    form_class = ContactForm

    def valid_submit(self, form):
        """
        This method is called whenever a form has been validated successfully and the␣
↪submit button has been pressed.
        If you don't implement this method, this method will not be called.
        This method can be used to do some extra work, such as sending an email in␣
↪this case.
        """
        pass
```

And the following urls configuration:

```
url(r'^contact/$', ContactFormView.as_view(), name="contact_form"),
```

And then in the template in which you are displaying the form, you should add:

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1/
↪jquery.js"></script>
{% load jquery_validation %}
{% include_validation %}
<script type="text/javascript">
    $(function()    {
        $('#form')
            .validate('{% url contact_form %}',
                      {
                          type: 'ul',
                          fields: ['id_email'],
                          event: 'focusout',
                          fieldSuccessCallback: function(field) {
                              if(field.next("span").length == 0) {
                                  field.after("<span class=\"ico-ok\">ok</span>");
                              }
                          }
                      });

    });
</script>
```

For ModelForms, the same pattern goes, except your inherit from AjaxModelFormView:

```
class ContactModelForm(forms.ModelForm):
    class Meta:
        model = Contact

class ContactView(AjaxModelFormView):
    template_name = "contact-model-form.html"
    form_class = ContactForm
    model = Contact

    def valid_submit(self, form):
        """
        Some goes here as for AjaxFormView
        """
```

AjaxModelFormView inherits from BaseCreateView, which saves the form through the defined model. This behaviour hasn't changed. There is no need to save your form in the valid_submit method.

As you can see, you need to have jQuery for this to work(here it is being loaded from google).

`$.validate`(*url*, *options*)

Arguments

- **url** (*string*) – The url to post to
- **options** (*dictionary*) – Dictionary of extra options

options

- *type* can either be ul, p or table. If no type is defined it defaults to ul.
- *fields* can be true if all fields need immediate ajax validation response or a list of fields that will need an immediate ajax response. In the example above only the email field will show imediate validation errors when focusing out of the field.
- *event* can by any valid jQuery event which is executed on each field (this depends ofcourse on how you've set the fields property).

- *fieldSuccessCallback(field)* is a callback function that is triggered when the field is valid. It takes one parameter the field that has been validated successfully. In the the example it adds an icon after the field. This can be used to add some extra information to the form that the field has been valid. For example add a green border around the field or a message saying the e-mail address is still available. Takes the field that has been marked as invalid as the only parameter.
- *formSuccessCallback* is a callback function that is triggered when the form is valid, after it has been submited. This callback should be implemented to for example redirect to user to a success page or just remove the form and show a message that the form has been submited successfully.
- *fieldInvalidCallback(field)* is a callback function that is triggered when a field is invalid. You could use this to remove style/elements you've added if the field has been marked as valid before. In other words if the fieldSuccessCallback function has been called on the field, before, and added some extra markup/style, that should be removed now. Takes the field that has been marked as invalid as only parameter.

The AjaxValidatingFormMixin only overwrites the post handler. So a normal GET request will serve the renderd form as defined in contact-form.html, as usual. POST request will validate the form and return a JSON response dictionary. If the form is invalid the dictionary will contain an other dictionary of errors, listing the fields that have been marked as invalid. You can implement one method form_is_valid, which is triggered from from_valid before rendering the json response. In this method you can do additional tasks, such as sending an email. The response will be handled by the jQuery plugin.

CHAPTER 2

Serving Ajax Validation With Your Static Media Server

By default, if you use the template tag included with Ajax Validation, the script will be placed inside HTML <script> tags and served up this way. This is usually fine for development, however for production it is reccomended you serve Ajax Validation with your seperate static media server.

To do this you should copy the `jquery-ajax-validation.js` file from `ajax_validation/media/ajax_validation/js/` and put it somewhere in the directory structure where the rest of your site's static media is. You should then replace all instances of `{% include_validation %}` with:

```
<script type="text/javascript" src="PATH_TO_FILE_HERE"></script>
```

where `PATH_TO_FILE_HERE` is the location your media server is serving the file from.

Project source: https://github.com/jonasgeiregat/django-ajax-forms

Index

Symbols