FREE eBook

LEARNING

django-forms

Free unaffiliated eBook created from
Stack Overflow contributors.

#django-forms

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: django-forms

It is an unofficial and free django-forms ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official django-forms.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with django-forms

Remarks

This section provides an overview of what django-forms is, and why a developer might want to use it.

It should also mention any large subjects within django-forms, and link out to the related topics. Since the Documentation for django-forms is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

Detailed instructions on getting django-forms set up or installed.

Read Getting started with django-forms online: https://riptutorial.com/django-forms/topic/8924/getting-started-with-django-forms

Chapter 2: Django Built-in forms

Introduction

Django is shipped with several views that require forms. These forms are, naturally, built-in. A good example are Authentication Built-in forms.

This topic intends to bring documentation on how to work with these forms.

Examples

Add custom CSS classes

Built-in forms are great but sometimes there is a need to customize them, adding new fields or simply changing CSS attributes.

This example is applicable to several use cases but here it is presented regarding PasswordChangeForm and its use in a Bootstrap website.

The solution is to create another Form that inerits `PasswordChangeForm` update the Widget:

```
class PasswordChangeCustomForm(PasswordChangeForm):
    def __init__(self, user, *args, **kwargs):
        super(PasswordChangeCustomForm, self).__init__(user,*args, **kwargs)
        for field in self.fields:
            self.fields[field].widget.attrs['class'] = 'form-control'
```

If you only pretend to change certain fields you may do:

```
class PasswordChangeCustomForm(PasswordChangeForm):
    def __init__(self, user, *args, **kwargs):
        super(PasswordChangeCustomForm, self).__init__(user, *args, **kwargs)
        self.fields['old_password'].widget.attrs.update({'class': 'form-control'})
        self.fields['new_password1'].widget.attrs.update({'class': 'form-control'})
        self.fields['new_password2'].widget.attrs.update({'class': 'form-control'})
```

Note: all bootstrap forms require the class `form-control` to keep the website look and feel.

Read Django Built-in forms online: https://riptutorial.com/django-forms/topic/8927/django-built-in-forms

Chapter 3: Testing

Introduction

One core feature of Django is unit tests.

This topic intends to bring a complete documentation on how to test forms.

Examples

Simple Test

```
from django.test import TestCase
from myapp.forms import MyForm

class MyAppTests(TestCase):
    def test_forms(self):
        form_data = {'field1': 'fieldvalue1'}
        form = MyForm(data=form_data)
        self.assertTrue(form.is_valid())
```

Read Testing online: https://riptutorial.com/django-forms/topic/8928/testing

Chapter 4: Using Model Form

Introduction

Django ModelForm enables the creation of a Form class from a Django model.

Examples

Using Django Model Form with Django Class Based View.

Django Model Form with Django Class Based view is a classic way of building pages to do create/update operations in django application quickly. Within the form we can put methods to execute tasks. Its a cleaner way to put tasks in forms rather than putting in views/models.

To give an example using Django Model Form, first we need to define our Model.

```
class MyModel(models.Model):
   name = models.CharField(
          verbose_name = 'Name',
          max_length = 255)
```

Now let us make a form using this model:

```
class MyModelForm(forms.ModelForm):

    class Meta:
        model = MyModel
        fields = '__all__'
```

Lets add a method to print hello world in it.

```
class MyModelForm(forms.ModelForm):
    class Meta:
        model = MyModel
        fields = '__all__'

    def print_hello_world(self):
         print('Hello World')
```

Lets make a template to display the form:

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

Now we will use this form in three different views which will respectively Create and Update tasks.

```
from django.views.generic.edit import CreateView, UpdateView
from myapp.models import MyModel

class MyModelCreate(CreateView):
    model = MyModel
    fields = ['name']
    form_class = MyModelForm
    template_name = 'my_template.html'

    def form_valid(self, form):
        # This method is called when valid form data has been POSTed.
        # It should return an HttpResponse.
        form.print_hello_world() # This method will print hello world in console
        return super(MyModelCreate, self).form_valid(form)


class MyModelUpdate(UpdateView):
    model = MyModel
    fields = ['name']
    form_class = MyModelForm
    template_name = 'my_template.html'
```

Now lets create a urls for accessing those views.

```
from django.conf.urls import url
from myapp.views import MyModelCreate, MyModelUpdate

urlpatterns = [
    # ...
    url(r'mymodel/add/$', MyModelCreate.as_view(), name='author-add'),
    url(r'mymodel/(?P<pk>[0-9]+)/$', MyModelUpdate.as_view(), name='author-update')
]
```

Okay, our work has been done. We can access url: `localhost:8000/mymodel/add` for creating entry in the model. Also access `localhost:8000/mymodel/1` to update that entry.

Making fields not editable

Django `1.9` added the Field.disabled attribute:

> The disabled boolean argument, when set to True, disables a form field using the disabled HTML attribute so that it won't be editable by users. Even if a user tampers with the field's value submitted to the server, it will be ignored in favor of the value from the form's initial data.

And so you only need to do:

```
MyChangeForm(ModelForm):

    def __init__(self, *args, **kwargs):
        super(MyChangeForm, self).__init__(*args, **kwargs)
        self.fields['<field_to_disable>'].disabled = True
```

And creating the form you need:

```
MyChangeForm(initial={'<field_to_disable>': "something"})
```

Before version `1.9` you had to:

```
class MyChangeForm(ModelForm):
    def __init__(self, *args, **kwargs):
        super(ItemForm, self).__init__(*args, **kwargs)
        instance = getattr(self, 'instance', None)
    if instance and instance.id:
        self.fields['<field_to_disable>'].required = False
        self.fields['<field_to_disable>'].widget.attrs['disabled'] = True

def clean_<field_to_disable>(self):
    # As shown in the above answer.
    instance = getattr(self, 'instance', None)
    if instance:
        return instance.<field_to_disable>
    else:
        return self.cleaned_data.get('<field_to_disable>', None)
```

And creating the form you need:

```
MyChangeForm(instance=MyChange.objects.get_or_create(<field_to_disable>="something"))
```

This example was based on this question.

Read Using Model Form online: https://riptutorial.com/django-forms/topic/8926/using-model-form

Credits

S. No	Chapters	Contributors
1	Getting started with django-forms	Community
2	Django Built-in forms	NBajanca
3	Testing	NBajanca
4	Using Model Form	NBajanca, ruddra