django_forms

February 15, 2018

1 Django Forms

1.1 Basic Example of setting up a Form Class:

- Looks very similar Model Class syntax.
- Widgets tell Django which type of HTML element it should use when rendering that field in the form.

```
from django import forms

class ExampleForm(forms.Form):
    ex1 = forms.CharField()
    ex2 = forms.CharField(widget=forms.Textarea)
    email = forms.EmailField()
    link = forms.URLField()
```

1.2 Creating an Example View for a Form:

```
from django.shortcuts import render
from django.core.urlresolvers import reverse
from django.http import HttpResponseRedirect
from django.contrib import messages

from . import forms

def suggestion_view(request):
    form = forms.<form_name>()
    if request.method == 'POST'
        form = forms.<form_name>(request.POST)
        if form.is_valid():
            messages.add_message(request, message.SUCCESS, "Thanks for your suggestion")
            return HttpResponseRedirect(reverse('suggestion')
    return render(request, '<form_html>.html', {'form': form})
```

1.3 Creating HTML Doc with a Django Form:

```
<form action="" method="POST">
    {{ form.as_p }}
```

```
    {% csrf_token %}
    <input type="submit">
</form>
```

1.4 Emails from Django

- In settings.py add the following:

```
EMAIL_BACKEND = 'django.core.mail.backends.filebased.EmailBackend'
EMAIL_FILE_PATH = os.path.join(BASE_DIR, 'suggestions')
```

- In views.py add the following:

```
from django.core.mail import send_mail

send_mail(
    "<subject>",
    "<body>",
    "<from_email>",
    "<to_email>",
)
```

Example using a form:

```
send_mail(
    "Suggestion from {}".format(form.cleaned_data['name']),
    form.cleaned_data['suggestion'],
    '{name} <{email}'.format(**form.cleaned_data),
    ['example@example.com'],
)
```

1.5 Custom Field Validation:

- Example of Honeypot field in a form (i.e. catching bots that autofill fields in an effort to spam you.)
- You can test this by adding a filled in `value` argument to the `input` html tag relating to this field.
- `clean_honeypot` method overrides the cleaning method typically implemented by Django. Django knows to run this instead because when the `is_valid()` method is run, it is looking for methods that start with `clean_` and end with the name of the field there meant to clean.
- More Info at: Django Docs: Cleaning Fields
- The following should be added to forms.py.

```
honeypot = forms.CharField(required=False,
                           widget=forms.HiddenInput,
                           label="Leave Empty")

def clean_honeypot(self):
    honeypot = self.cleaned_data['honeypot']
```

```
    if len(honeypot):
        raise forms.ValidationError(
            "honeypot should be left empty, BAD BOT!")
    return honeypot
```

1.6 Using Validators:

- Using Django's built-in validators

```
from django.core import validators

some_field = forms.CharField(validators=[validators.MaxLengthValidator(0)])
```

1.7 Creating Validators:

- Writing and Using Validators

```
from django.core import validators

def my_validator(value):
    if value:
        raise forms.ValidationError('is not empty')

# ......
another_field = forms.CharField(validators=[my_validator])
```

1.8 Cleaning (Checking Input) Entire Form:

- Cleaning entire form requires a method called `clean`, and may be filled with anything you like.
- `super().clean()` - Grabs all the data from the form.

```
email = forms.EmailField()
verify_email = forms.EmailField(help_text="Please verify your email")

def clean(self):
    cleaned_data = super().clean()
    email = cleaned_data.get('email')
    verify = cleaned_data.get('verify_email')

    if email != verify:
        raise forms.ValidationError(
            "Your passwords do not match")
```

1.9 Abstract Model Inheritance

- Creating Abstract models allows us to create a blueprint model for our models that share many fields in common to inherit from.

- An example might be an abstract class called user in which staff and student models inherit from.