Django Level Three

Time to level up your learning!

PIERIAN DATA

Django

- Welcome to Django Level Three!
- Hopefully you are now excited by the possibilities of the MTV workflows we've learned about!
- We are still missing a big piece to creating a full website - user input!

Django

- In this section we will be covering how to use Django Forms to accept User Input and connect it to the database and retrieve it later on.

Let’s get started!

Django - Basic Forms

Django Level Three

PIERIAN DATA

Django

- In this lecture we will conceptually walk through the process of creating a form with Django!
- We've covered Forms when discussing HTML, so why bother with Django Forms?
- What extra features do they bring?

Django

- Django Forms Advantages:
    - Quickly generate HTML form widgets
    - Validate data and process it into a Python data structure
    - Create form versions of our Models, quickly update models from Forms

Django

- The first thing we need to do is create a forms.py file inside the application!
- After that we call Django's built in forms classes (looks very similar to creating models).
- Let's see an example!

Django

- Example inside of forms.py:

```
from django import forms

class FormName(forms.Form):
    name = forms.CharField()
    email = forms.EmailField()
    text = forms.CharField(widget=forms.Textarea)
```

Django

- Note how similar this feels to creating a model!
- Now that we have the form created inside the application's forms.py file, we need to show it by using a view!

Django

- Inside our views.py file we need to import the forms (two ways to do this)
    - `from . import forms`
    - `from forms import FormName`

The . just indicates to import from the same directory as the current .py file

Django

- We can then create a new view for the form

```
def form_name_view(request):
    form = forms.FormName()
    return render(request,'form_name.html',
                          {'form':form})
```

Django

- Then we just add the view to the app's urls, either directly or with include(). Directly:

```
from basicapp import views
urlpatterns = [
        url(r'formpage/',views.form_name_view,
            name = 'form_name'),
]
```

Django

- We can then create the templates folder along with the html file that will hold the template tagging for the form.
- Remember to update the settings.py file to reflect the TEMPLATE_DIR variable!
- Also remember that your views should reflect subdirectories inside templates!

Django

- So now everything is setup for us to go into the form_name.html file inside templates/basicapp and add in the actual template tagging that will create the Django Form!

Django

- There are several ways you can “inject” the form using template tagging. You can just pass in the key from the context dictionary:
    - {{ form }}

Django

- Before we continue, let's have a quick side discussion about three topics:
    - HTTP
    - GET
    - POST

Django

- HTTP stands for Hypertext Transfer Protocol and is designed to enable communication between a client and a server.
- The client submits a request, the server then responds.

Django

- The most commonly used methods for this request/response protocol are GET and POST.
- GET - requests data from a resource
- POST - submits data to be process to a resource.

Django

- Those are the basics that we need to know for now, but you can check out the w3schools.com page on GET/POST for some more details, like what remains in browser history or what can be cached for future use.

Django

- Once you've put in the {{forms}} tag you should be able to see a very basic (and ugly) form on the page.
- However there is no <form> tag there.
- Let's look at what a more completed form html page would look like...

Django

- On your form_page.html

```
<div class="container">
  <form method="POST">
      {{ form.as_p }}
      {% csrf_token %}
      <input type="submit" class="btn btn-primary" value="Submit">
    </form>
</div>
```

Django

- Some added Bootstrap class styling calls

```
<div class="container">
  <form method="POST">
      {{ form.as_p }}
      {% csrf_token %}
      <input type="submit" class="btn btn-primary" value="Submit">
    </form>
</div>
```

Django

- Also calling form.as_p which uses <p>

```
<div class="container">
  <form method="POST">
      {{ form.as_p }}
      {% csrf_token %}
      <input type="submit" class="btn btn-primary" value="Submit">
    </form>
</div>
```

Django

- This gives it a nice format to work with.

```
<div class="container">
  <form method="POST">
      {{ form.as_p }}
      {% csrf_token %}
      <input type="submit" class="btn btn-primary" value="Submit">
    </form>
</div>
```

Django

- Check the Django docs for other

```
<div class="container">
  <form method="POST">
      {{ form.as_p }}
      {% csrf_token %}
      <input type="submit" class="btn btn-primary" value="Submit">
    </form>
</div>
```

Django

- Also added {% csrf_token %}

```
<div class="container">
  <form method="POST">
      {{ form.as_p }}
      {% csrf_token %}
      <input type="submit" class="btn btn-primary" value="Submit">
    </form>
</div>
```

Django

- This is the first time we've encountered thinking about site security measures!
- This is a Cross-Site Request Forgery (CSRF) token, which secures the HTTP POST action that is initiated on the subsequent submission of a form.

Django

- The Django framework requires the CSRF token to be present.
- If it is not there, your form may not work!
- It works by using a “hidden input” which is a random code and checking that it matches the user’s local site page.

Django

- You just need to remember to put the template tag there, you don't need to worry about the background.
- Now that we can show the form, let's discuss how to actually handle the form in a view!

Django

- Right now if we hit submit, nothing will happen.
- We need to inform the view that if we get a POST back, we should check if the data is valid and if so, grab that data.

Django

- We can do this by editing the view.
- We will talk a lot more about form validation later on, but upon receiving a validated form, we can access a dictionary like attribute of the “cleaned_data”.

Django

```
def form_name_view(request):
    form = forms.FormName()

    # Check to see if we get a POST back.
    if request.method == 'POST':
        # In which case we pass in that request.
        form = forms.FormName(request.POST)

        # Check to see form is valid
        if form.is_valid():
            # Do something.
            print("Form Validation Success. Prints in console.")
            print("Name"+form.cleaned_data['name'])
            print("Email"+form.cleaned_data['email'])
            print('Text'+form.cleaned_data['text'])
    return render(request,'basicapp/form_page.html',{'form':form})
```

Django

- Alright, we still have more topics to cover, like customizing form validation and connecting forms to a model!
- Let’s get some practice with what we know so far and create a basic form project and application from scratch!

Django

- Once we've done that, we'll revisit our original first_project and see how we can add a form that connects to a model!
- Let's get started!

Form Basics Code Along

Django Level Three

PIERIAN DATA

Form Validation

Django Level Three

PIERIAN DATA

Django

- In this lecture we will discuss hidden fields and how we can use them for custom field validation.
- The way our form is set up right now is pretty open to not only users, but potential “bots”.

Django

- Django has built-in validators you can conveniently use to validate your forms (or check for bots!)
- Everything we do here will be limited to the forms.py file, so we'll jump right into coding it all out!

Django

- We'll use the basicapp from the previous lecture and work with the following:
    - Adding a check for empty fields
    - Adding a check for a "bot"
    - Adding a clean method for the entire form.

Let’s get started!

Django Level Three

PIERIAN DATA

Model Forms

Django Level Three

PIERIAN DATA

Django

- We’ve seen how we can use Django Forms to grab information from the user and then do something with it.
- So far we’ve only printed out that information, but what if we wanted to save it to a model?

Django

- Luckily Django makes accepting form input and passing it to a model very simple!
- Instead of inheriting from the forms.Forms class, we will use forms.ModelForm in our forms.py file.

Django

- This helper class allows us to create a form from a pre-existing model
- We then add an inline class (something we haven’t seen before) called Meta
- This Meta class provides information connecting the model to the form.

Django

- Let’s see some example code of what this new type of ModelForm class would look like.

Django

- Example:

```
from django import forms
from myapp.models import MyModel
class MyNewForm(forms.ModelForm):
    # Form Fields go here
    class Meta:
        model = MyModel
        fields = # Let's see the options!
```

Django

- The fields attribute will connect to model

```
from django import forms
from myapp.models import MyModel
class MyNewForm(forms.ModelForm):
    # Form Fields go here
    class Meta:
        model = MyModel
        fields = # Let's see the options!
```

Django

- Many ways to make this connection!

```
from django import forms
from myapp.models import MyModel
class MyNewForm(forms.ModelForm):
    # Form Fields go here
    class Meta:
        model = MyModel
        fields = # Let's see the options!
```

Django

- Need to think about security for fields!

```
from django import forms
from myapp.models import MyModel
class MyNewForm(forms.ModelForm):
    # Form Fields go here
    class Meta:
        model = MyModel
        fields = # Let's see the options!
```

Django

- Very common to just use this:

```
from django import forms
from myapp.models import MyModel
class MyNewForm(forms.ModelForm):
    class Meta:
        model = MyModel
        fields = # Let's see the options!
```

Django

- Have the form be generated from model

```
from django import forms
from myapp.models import MyModel
class MyNewForm(forms.ModelForm):
    class Meta:
        model = MyModel
        fields = # Let's see the options!
```

Django

- This saves you work!

```
from django import forms
from myapp.models import MyModel
class MyNewForm(forms.ModelForm):
    class Meta:
        model = MyModel
        fields = # Let's see the options!
```

Django

- But if you want custom validators...

```
from django import forms
from myapp.models import MyModel
class MyNewForm(forms.ModelForm):
    class Meta:
        model = MyModel
        fields = # Let's see the options!
```

Django

- But if you want custom validators...

```
from django import forms
from myapp.models import MyModel
class MyNewForm(forms.ModelForm):
    # Form Fields go here with validators params
    class Meta:
        model = MyModel
        fields = # Let's see the options!
```

Django

- Option #1: Set it to “__all__”

```
from django import forms
from myapp.models import MyModel
class MyNewForm(forms.ModelForm):
    # Form Fields go here
    class Meta:
        model = MyModel
        fields = “__all__”
```

Django

- Option #2: exclude certain fields

```
from django import forms
from myapp.models import MyModel
class MyNewForm(forms.ModelForm):
    # Form Fields go here
    class Meta:
        model = MyModel
        exclude = [“field1”,“field2”]
```

Django

- Option #3: List included fields

```
from django import forms
from myapp.models import MyModel
class MyNewForm(forms.ModelForm):
    # Form Fields go here
    class Meta:
        model = MyModel
        fields = (“field1”,“field2”)
```

Django

- Check out the documentation for more discussion on connecting fields in the form to fields in the model.
- To get some practice with all of this, let's try adding a Model Form to our proTwo from Django Level Two!

Django

- This project had a single User class in its models, we will connect it to a form allowing users to register their names and emails to the site.
- This logic could easily be used to create a simple Coming Soon Landing Page!

Django

- To get started, make sure you have the ProTwo folder from the Django Level Two folder in the notes.
- To see the completed version of this, check the ProTwo folder in Django Level Three.

Django

- Let’s get started!

Model Forms - Exercise

Django Level Three

PIERIAN DATA

Django

- We will work with the ProTwo project folder from Django Level Two.
- Originally the user.html file used template tagging to display a list of all users.

Django

- We will change this to be a sign-up page.
- Connected to a ModelForm, the user will sign up on the user page and be taken back to the home page.
- A great exercise would be to try to do this on your own first!

Django

- (Optional) Exercise Steps:
    - Create a ModelForm in forms.py
    - Connect the form in the template
    - Edit views.py to show the form
    - Figure out how to .save() the data
    - Verify the model is admin registered

Django

- I highly encourage you to try it on your own! You will need to look at the documentation.
- Let’s get started!