Accelerating Information Technology

http://aiti.mit.edu

Ghana Summer 2012
Lecture DJ06–Django Forms
Forms

• How do we add data to the database?
  – admin interface
  – command line
  – forms (user-submitted)

• Forms are typically submitted using HTTP POST or GET protocols
Let’s look at HTML forms
1. Modify template so that it contains your form
Forms – template

```html
<html>
  <form action="" method=""POST">
    {{ form.as_p }}
    <input type=""submit"" value=""Submit!"">
  </form>
</html>
```

- We can render the form context variable a few different ways
- Try looking at the page source to see what HTML django is using behind the scenes in each case...
- Try `{{form.as_ul}}` instead
Forms step-by-step

1. Modify template so that it contains your form
2. Create your Form class in forms.py or views.py
Django Form class

class TextForm(forms.Form):
    text_message = forms.CharField()
    phone_number = forms.CharField()
Forms step-by-step

1. Modify template so that it contains your form
2. Create your Form class
3. Modify your View
Remember Http request?

```python
from django.http import HttpResponse

def hello(request):
    return HttpResponse("Hello world")
```

- HttpRequest has a lot of interesting functions
- Today, we care about: POST and GET
POST and GET

• Contain information submitted by the user
• “dictionary-like” objects
• GET = when you only want to display data
• POST = when you do other things as well, like change your database
Forms

- Each request has a POST and GET “dictionary” of parameters that were submitted using POST or GET

```html
<form action="/sms_interface/" method="POST">{% csrf_token %}
    <input type="text" name="text_message">
    <input type="text" name="phone_number">
</form>
```

These are values I might type into the form:

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>text_message</td>
<td>‘hello’</td>
</tr>
<tr>
<td>phone_number</td>
<td>0784751342</td>
</tr>
</tbody>
</table>
POST data

• Using the request.POST dictionary, we can access the attributes we want to use...

def mirror_response(request):
    if request.method == "POST":
        text_string = request.POST["text_message"]
        phone_number = request.POST["phone_number"]
        return HttpResponse('%s sent the text message %s')
    else:
        return HttpResponse("This is not a helpful way to handle non-POST requests")
Django Form class

- The **Form** class can help us out...

```python
class TextForm(forms.Form):
    text_message = forms.CharField()
    phone_number = forms.CharField()

def sms_handler(request):
    if request.method == "POST":
        text_info = TextForm(request.POST)
        if text_info.is_valid():
            form_data = text_info.cleaned_data
            text_body = text_info['text_message']
            phone_number = text_info['phone_number']
            return HttpResponse("%s sent %s" %
                (phone_number,text_body))
        else:
            my_rc = RequestContext(request,{
                'form':TextForm()})
            render_to_response('blank_window.html',my_rc)
```
Forms and Models

• What if we want to let users add data to our database?
• Add a book
• Add a comment to our blog
• Remember that you already have your model defined, now you want a way to represent that model through a form
We want this (but hopefully prettier)
We want this (but hopefully prettier)
Remember Movie example?

class Movie(models.Model):
    rating = models.IntegerField()
    title = models.CharField(max_length=100)
    genre = models.CharField()
    lead_actor = models.ForeignKey(Actor,related_name='lead_actor')
    support_actors = models.ManyToManyField(Actor,related_name='support')
We want this:

ForeignKey and ManyToMany fields: Dropdown and multiple select menus
class MovieForm(forms.Form):
    title = forms.CharField()
    genre = forms.CharField()
    rating = forms.IntegerField()
    # what should we do for lead actor and supporting actors?
    lead_name = forms.CharField()
    support_names = forms.CharField()

How do we create a Movie instance now and put it in our database?
def get_movie_data(request):
    if request.method == "POST":
        movie_form = MovieForm(request.POST)
        my_movie = Movie(title=movie_form.title, rating=movie_form.rating, genre = movie_form.genre)
        lead_actor = Actor.objects.get(name=movie_form.name)
        all_support_names = movie.support_names.split(",")
        my_movie.save()
        for some_name in all_support_names:
            my_movie.supporting_actors.add(Actor.objects.get (name=some_name))
        my_movie.save()
Bad solution

• Advantages:
  – Exercise our QuerySet API Skills

• Disadvantages:
  – That was miserable
ModelForm Class

Let’s create a form based on our Movie model

```python
from django.forms import ModelForm
from models import Movie
class MovieForm(ModelForm):
    class Meta:
        model = Movie
```
ModelForm Class

• One view function for two cases:
  – the user has submitted the form
  – the user wants to fill out the form

```python
def get_movie_data(request):
    if request.method == "POST":
        movie_form = MovieForm(request.POST)
        my_movie = movie_form.save()
        return HttpResponse("The movie %s was successfully entered in the database")
    else:
        my_form = MovieForm()
        my_rc = RequestContext(request, {'form': my_form})
        return render_to_response('movie_app/movie_form.html', my_rc)
```
ModelForm Class

- Django does a ridiculous amount of HTML work on our behalf

ForeignKey and ManyToMany fields: Dropdown and multiple select menus.