

Django Forms

COMP 8347

Slides prepared by Dr. Arunita Jaekel
arunita@uwindSOR.ca



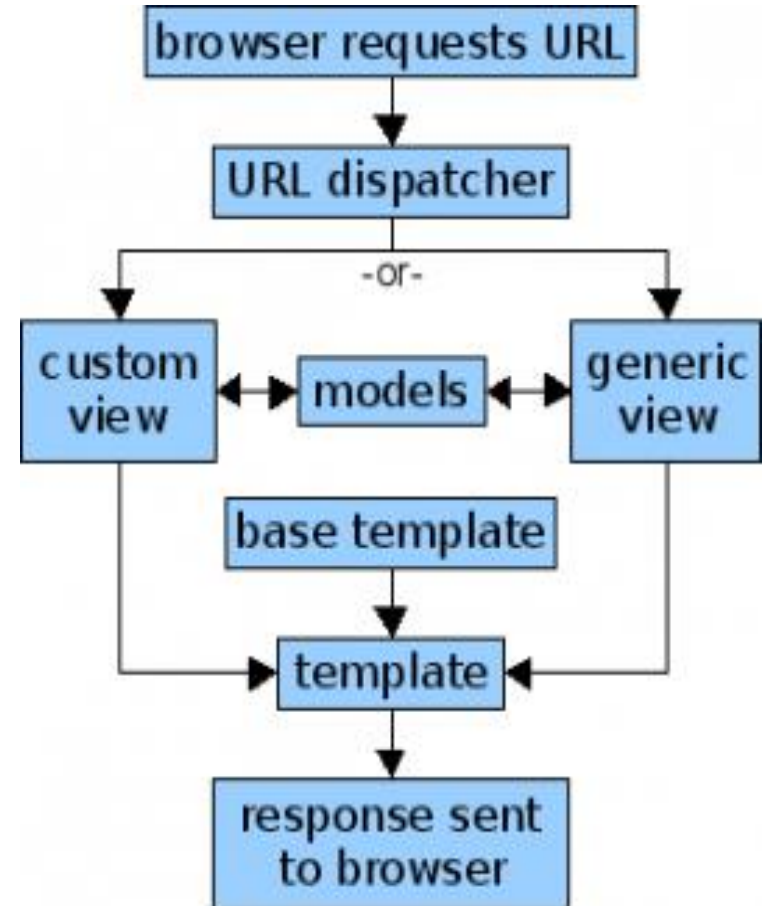
Django Forms

- Topics
 - Django Forms
 - The Form Class
 - Fields and Widgets
 - Rendering Forms
 - Validating Forms
 - ModelForm



Review MTV Architecture

- Represent data organization; defines a table in a database.
- Contain information to be sent to client; help generate final HTML.
- Actions performed by server to generate data.



www.tikalk.com/files/intro-to-django.ppt



HTML Forms

- *Form*: A collection of elements inside `<form>...</form>`
 - allow user to enter text, select options, manipulate objects etc.
 - send information back to the server.
- In addition to `<input>` elements, a form must specify:
 - *where*: the URL to which the data corresponding to the user's input should be returned
 - *how*: the HTTP method to be used to return data.
 - `<form action="/your-name/" method="post">`



GET and POST

- **GET**: bundles the submitted data into a string, and uses this to compose a URL.
 - The URL contains the address where the data must be sent, as well as the data keys and values.
- **POST**: Form data is transmitted in body of request, not in URL.
 - Any request that could be used to change the state of the system should use POST method.



GET and POST

- GET should be used **only** for requests that **do not** affect the state of the system.
 - Not suitable for large quantities of data, or for binary data, such as an image.
 - Unsuitable for a password form, because the password would appear in the URL.
 - GET is suitable for things like a web search form
 - the URLs that represent a GET request can easily be bookmarked, shared, or resubmitted



Django's Functionality

- Form processing involves many tasks:
 - Example: prepare data for display, render HTML, validate data, and save data
- Django can automate and simplify much of this work.
Handles 3 main areas:
 - preparing and restructuring data ready for rendering
 - creating HTML forms for the data
 - receiving and processing submitted forms and data from the client



Building a Form

- Sample HTML form, to input your name.

```
<form action="/inp/" method="post">  
  <label for="your_name">Username: </label>  
  <input id="your_name" type="text" name="your_name"  
    maxlength="100" >  
  <input type="submit" value="OK">  
</form>
```

- Components:
 - Data returned to URL **/inp/** using **POST**
 - Text field labeled **Username:**
 - Button marked **“OK”**



Building a Django Form

- Create a Form subclass:

```
from django import forms
```

```
class NameForm(forms.Form):
```

```
    your_name = forms.CharField(max_length=100)
```

- This defines a Form class with a single field (your_name).

- Creates a text input field
- Associates a label with this field
- Sets a maximum length of 100 for the input field.

- When rendered it will create the following HTML

```
<label for="id_your_name">Your name: </label>
```

```
<input id="id_your_name" type="text" name="your_name"  
      maxlength="100" >
```

- NOTE: It does not include `<form>` `</form>` tags or `submit` button.



Field Arguments

- **Field.required:** By default, each Field class assumes the value is required
 - empty value raises a **ValidationError** exception
- **Field.label:** Specify the “human-friendly” label for this field.
name = forms.CharField(label='Your name')
- **Field.initial:** Specify initial value to use when rendering this Field in an unbound Form.
name = forms.CharField(initial='John')
- **Field.widget:** Specify a Widget class to use when rendering this Field.
- **Field.error_messages:** Override the default messages that the field will raise.
 - Pass in a dictionary with keys matching the error messages you want to override.
name = forms.CharField(error_messages={'required': 'Please enter your name'})
 - ...
 - ValidationError: [u'Please enter your name']
 - The default error message is: [u'This field is required.']



Widgets

- Each form field has associated **Widget** class
 - Corresponds to an HTML input element, such as **<input type="text">**.
 - Handles rendering of the HTML
 - Handles extraction of data from a GET/POST dictionary
 - Each field has a sensible **default** widget.
- Example: **CharField** has default **TextInput** widget → produces an **<input type="text">** in the HTML.
- **BooleanField** is represented by **<input type="checkbox">**
- You can **override** the default widget for a field.
- **BooleanField**: Default widget: **CheckboxInput**; Empty value: False
- **CharField**: Default widget: **TextInput**; Empty value: **''** (empty string)
- **ChoiceField**: Default widget: **Select**; Empty value: **''** (empty string)
- **EmailField**: Default widget: **EmailInput**; Empty value: **''** (empty string).
- **IntegerField**: Default widget: **TextInput** (typically); Empty value: None
- **MultipleChoiceField**: Default widget: **SelectMultiple**; Empty value: **[]** (empty list).



Create ContactForm Class

- **Create** forms in your app's *forms.py* file.
- **Instantiate** the form in your app's *views.py* file;
 - In view function corresponding to URL where form to be published
- **Render** the form by passing it as context to a template.
- Consider a form with four fields:
 - **subject, message, sender, cc_myself.**
 - Each field has an associated **field type**.
 - Example: CharField, EmailField and BooleanField

```
from django import forms
```

```
class ContactForm(forms.Form):
```

```
    subject = forms.CharField(max_length=100)
```

```
    message = forms.CharField(widget=forms.Textarea)
```

```
    sender = forms.EmailField()
```

```
    cc_myself = forms.BooleanField(required=False)
```



Instantiate and Render a Form

- Steps in rendering an object:
 1. **retrieve** it from the database in the view
 2. pass it to the template **context**
 3. create HTML using template variables
- Rendering a form is similar
- When dealing with a form we typically **instantiate** it in the view.
 - process form if needed
 - Render the form:
 - pass it to the template **context**
 - create HTML using template variables



Bound and Unbound Forms

- A Form instance can be i) **bound** to a set of data, or ii) **unbound**.
 - **is_bound()** method will tell you whether a form has data bound to it or not.
- An **unbound** form has no data associated with it.
 - When rendered, it will be empty or contain default values.
 - To create simply instantiate the class. e.g. **f = NameForm()**
- A **bound** form has submitted data,
 - Can render the form as HTML with the data displayed in the HTML.
 - To bind data to a form: Pass the data as a dictionary as the first parameter to your Form class constructor
 - The **keys** are the field names, correspond to the **attributes** in Form class.
 - The **values** are the data you're trying to validate.

```
data = {'your_name': 'Arunita'}  
form = NameForm (data)
```

or

```
form = ContactForm(request.POST)
```



The View

- Data sent back typically processed by same view which published the form
 - If form is submitted using **POST**: populate it with data submitted
 - If arrived at view with **GET** request: create an **empty form instance**;

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
from myapp.forms import ContactForm
```

```
def contact(request):
    if request.method == 'POST': # if POST process submitted data

        # create a form instance; populate with data from the request:
        form = ContactForm(request.POST)
        if form.is_valid(): # check whether it's valid:
            # process the data in form
            # ... return render(request, 'response.html', {'myform': form})
    else:
        form = ContactForm() # if a GET create a blank form
        return render(request, 'contact.html', {'myform': form})
```



A Sample Template

- Get your form into a **template**, using the **context**.
 - **return** render(request, 'contact.html', {'myform': form})
- If the form is called '**myform**' in the context, use **{{myform}}** in template.
- NOTE: This will **not** render the **<form>** tags or the **submit** button
- The form can be rendered manually or using one of the options:
 - **form.as_table**, **form.as_p** or **form.as_ul**

```
<form action="/your-name/" method="post">
    {% csrf_token %}
    {{ myform }}
    <input type="submit" value="Submit" />
</form>
```

- The form's fields and their attributes will be unpacked into HTML markup from the **{{ myform }}** form variable.
- The **csrf_token** template tag provides an easy-to-use protection against Cross Site Request Forgeries



Rendering Options

```
from django import forms
class ContactForm(forms.Form):
    subject =
    forms.CharField(max_length=100)
    message =
    forms.CharField(widget=forms.Textarea)
    sender = forms.EmailField()
    cc_myself =
    forms.BooleanField(required=False)
```

- The name for each tag is from its **attribute name**.
 - The text label for each field is generated from the field name by converting all underscores to spaces and upper-casing the first letter. Default suffix is **‘:’**
 - Example: `cc_myself` → **‘Cc myself:’**
 - These are defaults; you can also specify labels manually.
- Each text label is surrounded in an HTML **`<label>`** tag, which points to a form field via its **id**.
 - Its id is generated by prepending **‘id_’** to the field name.
- The **id** attributes and **`<label>`** tags are included in the output by default.
 - To change this, **set `auto_id=False`**



Rendering Forms

- Output of {{myform.as_p}}

```
<form action="/myapp/contact/" method="post">
```

```
<p><label for="id_subject">Subject:</label>
```

```
<input id="id_subject" type="text" name="subject"
```

```
maxlength="100" /></p>
```

```
<p><label for="id_message">Message:</label>
```

```
<input type="text" name="message" id=
```

```
"id_message" /></p>
```

```
<p><label for="id_sender">Sender:</label>
```

```
<input type="email" name="sender"
```

```
id="id_sender"/></p>
```

```
<p><label for="id_cc_myself">Cc myself:</label>
```

```
<input type="checkbox" name="cc_myself"
```

```
id="id_cc_myself" /></p>
```

```
<input type="submit" value="Enter Contact Info" />
```

```
</form>
```

```
from django import forms
```

```
Class ContactForm(forms.Form):
```

```
    subject =
```

```
    forms.CharField(max_length=100)
```

```
    message =
```

```
    forms.CharField(widget=forms.Textarea)
```

```
    sender = forms.EmailField()
```

```
    cc_myself =
```

```
    forms.BooleanField(required=False)
```



Form Validation

- ***Form.is_valid()*** : A method used to validate form data.
 - **bound** form: runs validation and returns a boolean (**True** or **False**) designating whether the data was valid. Generates **myform.errors** attribute.
 - **unbound** form: always returns **False**; **myform.errors** = { }
- The validated form data will be in the **myform.cleaned_data** dictionary.
 - includes a key-value for **all** fields; even if the data didn't include a value for some optional fields.
 - Data converted to appropriate Python types
 - Example: **IntegerField** and **FloatField** convert values to Python int and float respectively.



Validated Field Data

```
>>> data = {'subject': 'hello',  
            'message': 'Hi there',  
            'sender': 'foo@example.com',  
            'cc_myself': True}  
>>> myform = ContactForm(data)  
>>> myform.is_valid()  
True  
>>> myform.cleaned_data  
{'cc_myself': True,  
  'message': u'Hi there',  
  'sender': u'foo@example.com',  
  'subject': u'hello'}
```

- The values in **cleaned_data** can be assigned to variables and used in the view function.

```
if myform.is_valid():  
    subj= myform.cleaned_data['subject']  
    msg= myform.cleaned_data['message']  
    sender = myform.cleaned_data['sender']  
    cc = myform.cleaned_data['cc_myself']  
  
    return HttpResponseRedirect('/thanks/')
```



Form Validation – if Errors Found

- ◉ Django automatically displays suitable error messages.
- **f.errors:** An attribute consisting of a **dict** of error messages.
form's data validated first time either you call **is_valid()** or access **errors** attribute.
- **f.non_field_errors():** A method that returns the list of errors from **f.errors** not associated with a particular field.
- **f.name_of_field.errors:** a list of form errors for a specific field, rendered as an unordered list.
E.g. `form.sender.errors()` → `[u'Enter a valid email address.']`



Displaying Errors

- Rendering a *bound* **Form** object automatically runs the form's validation
 - HTML output includes validation errors as a `<ul class="errorlist">` near the field.
 - The particular positioning of the error messages depends on the output method.

```
>>> data = {'subject': '', 'message': 'Hi there', 'sender': 'invalid email format',  
            'cc_myself': True}  
>>> f = ContactForm(data, auto_id=False)  
>>> print(f.as_p ())
```

```
<p><ul class="errorlist"><li>This field is required.</li></ul></p>
```

```
<p>Subject: <input type="text" name="subject" maxlength="100" /></p>
```

```
<p>Message: <input type="text" name="message" value="Hi there" /></p>
```

```
<p><ul class="errorlist"><li>Enter a valid email address.</li></ul></p>
```

```
<p>Sender: <input type="email" name="sender" value="invalid email address"  
/></p>
```

```
<p>Cc myself: <input checked="checked" type="checkbox" name="cc_myself"  
/></p>
```



ModelForm

- **ModelForm**: a helper class to create a **Form** class from a Django **Model**.
 - The generated **Form** class will have a **form field** for every **model field**
 - the order specified in the **fields** attribute.
 - Each model field has a corresponding default form field.
 - Example: **CharField** on model → **CharField** on form.
 - **ForeignKey** represented by **ModelChoiceField**: a **ChoiceField** whose choices are a model QuerySet.
 - **ManyToManyField** represented by **ModelMultipleChoiceField**: a **MultipleChoiceField** whose choices are a model QuerySet.
 - If the model field has **blank=True**, then **required = False** .
 - The field's **label** is set to the **verbose_name** of the model field, with the first character capitalized.
 - If the model field has **choices** set, then the form field's **widget** will be set to **Select**, with choices coming from the model field's choices.



ModelForm Example

```
class Book(models.Model):  
    title = models.CharField(max_length=100)  
    length = models.IntegerField()  
    pub_date = models.DateField()
```

```
from django.forms import ModelForm  
from myapp.models import Book
```

```
# Create the form class.
```

```
class BookForm(ModelForm):  
    class Meta:  
        model = Book  
        fields = ['title', 'pub_date', 'length']
```

```
# Creating a form to add a book  
form = BookForm()
```

```
# Create form to change book in db.  
book = Book.objects.get(pk=1)  
form = BookForm(instance=book)
```



ModelForm Example

```
from django.db import models
```

```
PROV_CHOICES = ( ('ON', 'Ontario.'), ('AB',  
    'Alberta.'), ('QC', 'Quebec.'), )
```

```
class Author(models.Model):
```

```
    name =
```

```
        models.CharField(max_length=100)
```

```
    prov = models.CharField(max_length=3,  
        choices=PROV_CHOICES)
```

```
    birth_date =
```

```
        models.DateField(blank=True, null=True)
```

```
class Book(models.Model):
```

```
    title = models.CharField(max_length=100)
```

```
    authors =models.ManyToManyField(Author)
```

```
from django.forms import  
    ModelForm
```

```
class AuthorForm(ModelForm):
```

```
    class Meta:
```

```
        model = Author
```

```
        fields = ['name', 'prov',  
            'birth_date']
```

```
class BookForm(ModelForm):
```

```
    class Meta:
```

```
        model = Book
```

```
        fields = ['title', 'authors']
```



Form vs ModelForm Examples

```
from django.forms import
    ModelForm
class AuthorForm(ModelForm):
    class Meta:
        model = Author
        fields = ['name', 'prov',
            'birth_date']

class BookForm(ModelForm):
    class Meta:
        model = Book
        fields = ['title', 'authors']
```

```
from django import forms
class AuthorForm(forms.Form):
    name =
        forms.CharField(max_length=100)
    prov = forms.CharField(max_length=3,
        widget=forms.Select(choices=PROV
            _CHOICES))
    birth_date =
        forms.DateField(required=False)

class BookForm(forms.Form):
    title =
        forms.CharField(max_length=100)
    authors =
        forms.ModelMultipleChoiceField(
            queryset=Author.objects.all())
```



save() Method

- ***save() method:*** This method creates and saves a database object from the data bound to the form.
 - can accept an existing model instance as the keyword argument **instance**.
 - If this is supplied, **save()** will update that instance.
 - Otherwise, **save()** will create a new instance of the specified model
 - accepts an optional **commit** keyword argument (either True or False); **commit=True** by default.
 - If **commit=False**, then it will return an **object** that hasn't yet been saved to the database.
 - In this case, it's up to you to call **save()** on the resulting **model** instance.



ModelForm Validation

- Validation is triggered
 - implicitly when calling `is_valid()` or accessing the `errors` attribute
- Calling `save()` method can trigger validation, by accessing errors attribute
 - A `ValueError` is raised if `form.errors` is `True`.



Summary

- Django Forms
 - The Form Class
 - Fields and Widgets
- Rendering Forms
 - Validating Forms
 - Error messages
- ModelForm
 - Saving and validating ModelForms



- [1] <https://docs.djangoproject.com/en/3.0/topics/forms/>
- [2] <https://djangobook.com/django-tutorials/mastering-django-forms/>

