Django Forms

COMP 8347

Usama Mir

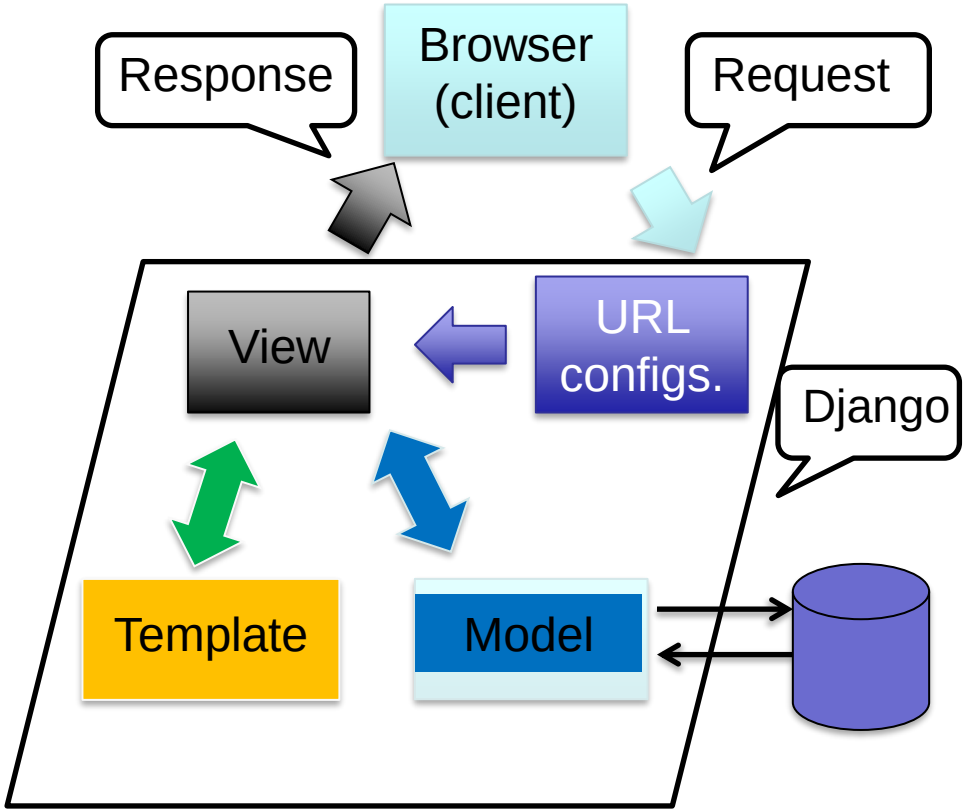
Usama.Mir@uwindsor.ca

Topics

- HTML Forms
- GET vs. POST
- Building a Basic Form in Django
- Rendering a Basic Django Form
- Form Validation
- Model Forms
- Save Method

Review: The MTV Architecture

- Represents data organization; defines a table in a database.
- Contains information to be sent to client; helps generate final HTML.
- Actions performed by server to generate data.



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.

The Form Class

- ***Form class***: describes a form and determines how it works and appears.
  - Like how a model describes the logical structure of an object
- ***Form Field***: a form class's fields map to HTML form <input> elements.
  - Form's fields are themselves classes;
  - Fields manage form data and perform validation when a form is submitted.
  - A form field is represented in the browser as a HTML "widget"

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'})
  - 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).

Changing a Widget

```
class PersonForm(forms.Form):
    first = forms.CharField(max_length=100, required=True)
    last = forms.CharField(max_length=100, required=True)
    middle = forms.CharField(max_length=100,
        widget=forms.TextInput(attrs={'size': 3}
    )
```

Use of that form would result in a `middle` field such as:

```
<input id="id_middle" maxlength="100" type="text" name="middle" size="3" />
```

Create ContactForm Class

- **Create** forms in your app's ***forms.py*** file.
- 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** 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.

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': 'Usama'}

      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**

Ex.

```
f = ContactForm(auto_id=False)

print(f.as_table())
```

Rendering Forms

```
from django import forms
Class ContactForm(forms.Form):
  subject =
  forms.CharField(max_length=1
  00)
  message =
  forms.CharField(widget=forms.
  Textarea)
  sender = forms.EmailField()
  cc_myself =
  forms.BooleanField(required=F
  alse)
```

- Output of {{myform.as_p}}

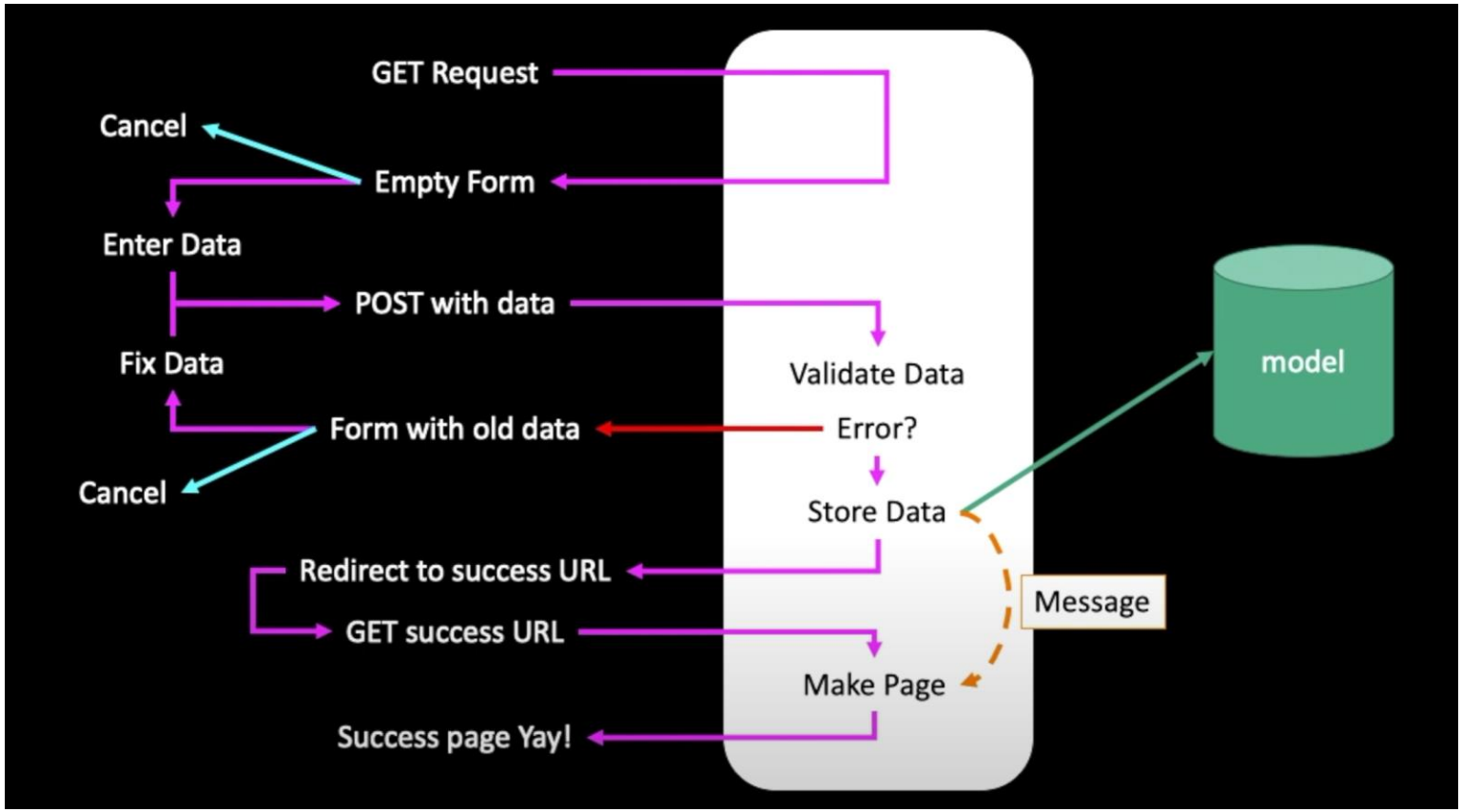
```
<form action="/myapp/contact/" method="post">
  <p><label for="id_subject">Subject:</label>
    <input id="id_subject" type="text“ name="subject"
        maxlength="100" /></p>
  <p><label for="id_message">Message:</label>
    <input id= "id_message" type="text"
        name="message" /></p>
  <p><label for="id_sender">Sender:</label>
    <input id="id_sender“ type="email" name="sender"
        /></p>
  <p><label for="id_cc_myself">Cc myself:</label>
    <input id="id_cc_myself" type="checkbox"
        name="cc_myself" /></p>
<input type="submit" value="Enter Contact Info" />
</form>
```

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.
  - unbound form: always returns **False**;

- The validated form data will be in the ContactForm.cleaned_data dictionary, where ContactForm is the name of the form.
  - includes a key-value for ***all*** fields; even if the data didn't include a value for some optional fields.
  - *https://www.javatpoint.com/django-form-validation*

Form Validation Process



ModelForms

- Refer to the example we do in the class.

save() Method

- ***save() method:*** This method creates and saves a database object from the data bound to the form.
- To save changes to an object that's already in the database, use save(). Ex. item.save()
- This performs an UPDATE SQL statement behind the scenes.
- Django does not hit the database until you explicitly call save().

References

- https://docs.djangoproject.com/en/5.0/ref/forms/api/
- https://docs.djangoproject.com/en/5.0/ref/forms/fields/
- https://docs.djangoproject.com/en/5.0/ref/forms/widgets/#:~:text=A%20widget%20is%20Django's%20representation,targeting%20.
- https://www.youtube.com/watch?v=kRXiMIFKzoQ
- Python Web Development with Django, by J. Forcier et al.
- Slides from Dr. Arunita and Dr. Saja