



University of Windsor

Django Templates

COMP 8347

Usama Mir

usamamir@uwindor.ca

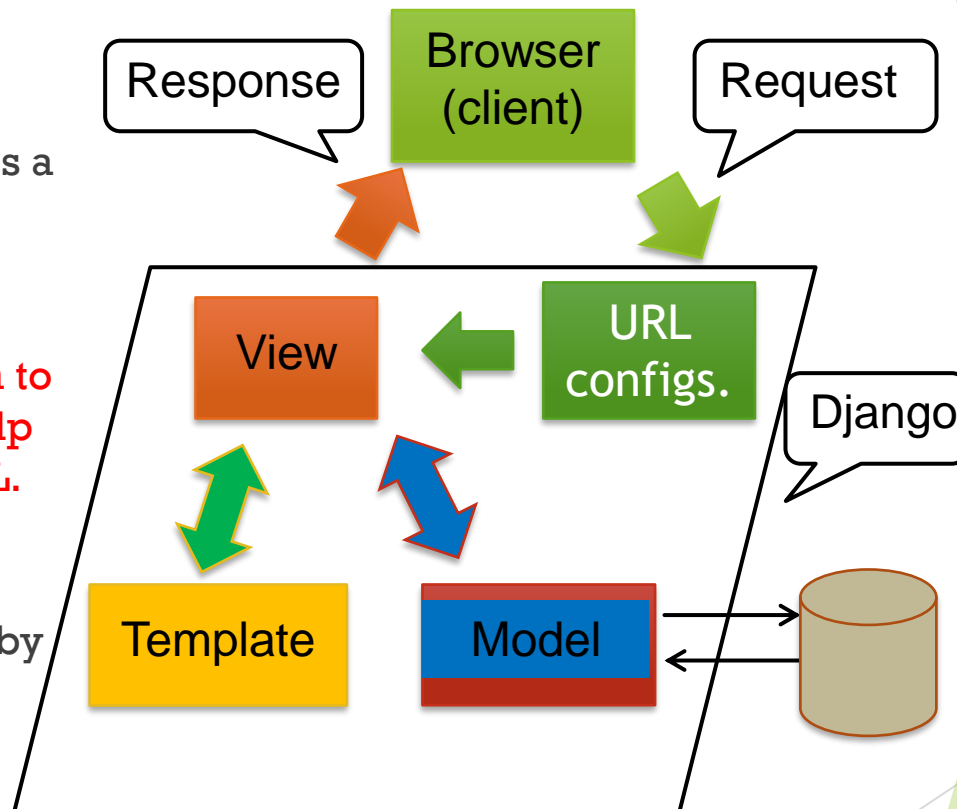
Django Templates

Topics

- ▶ Introduction to Templates
- ▶ Django Template Language
- ▶ Template Inheritance
- ▶ Including Static Files in Templates

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.



Templates

- ▶ **Template:** a text document, or a normal Python string, that is marked-up using the Django template language.
 - ▶ Contains **static** content (e.g. HTML) and **dynamic** mark-up (specifying logic, looping, etc.)
 - ▶ Can contain **block tags** and/or **variables**
 - ▶ Choice of which template to use and data to display is made in **view function** itself (or through its arguments).

Shortcut Functions

- ▶ *django.shortcuts*: A package that collects helper functions and classes.
 - ▶ “span” multiple levels of MTV.
 - ▶ these functions/classes introduce controlled coupling for convenience.
- ***render()***: A Django shortcut function
 - Combines a given template with a given context dict and returns an **HttpResponse** object with that rendered text.

Render()

- ▶ `render(request, template_name, context=None, content_type=None, status=None, using=None)`
 - ▶ Required arguments
 - ▶ *request*: request obj used to generate the response.
 - ▶ *template_name*: The full name of a template to use or sequence of template names.
 - ▶ *context*: a *dict-like* object used for passing information to a template.
 - ▶ Every rendering of a template typically requires a context.
 - ▶ Default (i.e., empty dict) - would not be very dynamic.
 - ▶ Contains a dictionary of 'key-value' pairs.
 - ▶ *content_type*: The MIME type to use for the resulting document. Defaults to 'text/html'.
 - ▶ *status*: The status code for the response. Default is 200.
 - ▶ *using*: The NAME of a template engine to use for loading the template. Ex. BACKEND, DIR, APP_DIR etc
 - ▶ www.webforefront.com/django/1.11/customizedjangotemplates.html

An Example

```
from myapp.models import Book
from django.http import HttpResponse
def my_view(request):
    # View code here...
    books = Book.objects.all()
    response=HttpResponse()
    for item in books:
        para= '<p>' + item.title+ '</p>'
        response.write(para)
    return response
```

```
from django.shortcuts import render
from myapp.models import Book
def my_view(request):
    # View code here...
    books = Book.objects.all()
    return render(request,
        'myapp/index.html', { 'books': books
        })
```

Template Language Syntax

- ▶ The Django template system is meant to express presentation, not program logic.
 - ▶ The language does not try to be (X)HTML compliant.
 - ▶ Contains **variables** and **tags**.
 - ▶ **Variables**: get replaced with **values** when the template is evaluated.
 - ▶ look like this: `{{ variable }}`.
 - ▶ **Tags**: control the **logic** of the template.
 - ▶ Look like this: `{% tag %} ... tag contents ... {% endtag %}`

Variables

- ▶ When the template engine encounters a variable `{{variable}}`
 - ▶ it **evaluates** that variable and replaces it with the result.
- ▶ *Variable names*: any combination of alphanumeric characters and underscore ("_").
 - ▶ Cannot start with underscore
 - ▶ cannot have spaces or punctuation characters
 - ▶ The dot (".") has a special meaning

The Dot-lookup Syntax

- ▶ When the template system encounters a dot (.) e.g. `{{my_var.x}}`:
 - ▶ Tries the following lookups, in this order:
 - ▶ Dictionary lookup
 - ▶ Attribute or method lookup
 - ▶ Numeric index lookup
- ▶ Example: `<h1>{{ employee.age}}</h1>`
 - ▶ Will be replaced with the `age` attribute of `employee` object

Filters

- ▶ *Filters*: Allow you to modify context variables for display.
 - ▶ Similar to unix **pipes** (`|`), e.g. `{{ name|lower }}`
 - ▶ Can be “**chained**” `{{ text|escape|linebreaks }}`
 - ▶ The output of one filter is applied to the next.
 - ▶ Some filters take arguments.
 - ▶ `{{ story|truncatewords:50 }}`: displays 1st 50 words of story variable.
 - ▶ arguments that contain spaces must be quoted
 - ▶ `{{ list|join:", " }}`: joins a list with comma and space

Tags

- ▶ Tags `{% tag %}` can have different functionality.
 - ▶ e.g. control flow, loops, logic
 - ▶ may require beginning and ending tags
 - ▶ some useful tags:
 - ▶ `for`
 - ▶ `if, elif, else`
 - ▶ `block and extends`

for Tag

- ▶ Used to loop over each item in an array
- ▶ Example:

```
<body>  
{% for book in booklist %}  
  <li>{{ book.title }}</li>  
{% endfor %}  
</body>
```

if, elif, else Tags

- Evaluates a variable

- if the variable is “true”, then the contents of the block are displayed

```
{% if my_list|length > 5 %}
```

```
<p> Number of selected items: {{ my_list|length }} </p>
```

```
{% elif my_list %}
```

```
<p> Only a few items were selected </p>
```

```
{% else %}
```

```
<p> {{my_list|default: 'Nothing selected.'}} </p>
```

```
{% endif %}
```

url Tag

- ▶ *url Tag*: Returns an **absolute path reference** (a URL without the domain name) matching a given view function.
 - ▶ may have **optional parameters** v1 v2 etc
 - ▶ All arguments required by the URLconf should be present.
 - ▶ `{% url 'path.to.some_view' v1 v2 %}`
 - ▶ `{% url 'path.to.some_view' arg1=v1 arg2=v2 %}`

Removing Hardcoded URLs

```
urlpatterns = [  
    path(<int:emp_id>/, views.detail, name='detail'),  
]
```

Matching url: **myapp/5/**

A hardcoded link in template file:

- ▶ `{{ author.name }}`
 - ▶ hard to change URLs on projects with many templates
 - ▶ Solution: use the `{% url %}` template tag, if name argument is defined in the corresponding *urls.py*
- ▶ `{{author.name }}`
 - ▶ looks up URL definition from the *myapp.urls* module
 - ▶ `path('<int:author_id>/', views.detail, name='detail')`
- ▶ If you want to change the URL
 - ▶ Matching url: **myapp/5/** → **myapp/emp_info/5/**
 - ▶ `path('emp_info/<int:emp_id>/', views.detail, name='detail')`
 - ▶ Don't need to change anything in template file

Namespacing URL Names

- ▶ Adding namespaces allows Django to distinguish between views with same names in different APPs.
 - ▶ add namespace in app level *urls.py* (after **import** instructions)
 - ▶ `app_name = 'myapp2'`
 - ▶ URL definition from the `myapp2.urls` module
 - ▶ `path('<int:author_id>/', views.detail, name='detail')`
 - ▶ In template file, refer to it as
 - ▶ `{{author.name}}`

1. [Toyota](#)
2. [Nissan](#)
3. [Ford](#)
4. [Honda](#)
5. [Mercedes](#)
6. [Cadillac](#)

Template Inheritance

- ▶ *Template inheritance*: allows you to build a base “skeleton” template that contains all the common elements of your site.
 - ▶ defines **blocks** that child templates can override
 - ▶ *block Tag*: Used in **base template** to define blocks that can be overridden by child templates.
 - ▶ tells template engine that a child template may override those portions of the template
 - ▶ `<title>{% block title %}Hello World{% endblock %}</title>`
 - ▶ *extends Tag*: Used in **child template**
 - ▶ tells the template engine that this template “**extends**” another template.

Base&Child Templates

```
<!DOCTYPE html>
<html lang="en">
<head>
  <link rel="stylesheet" href="style.css" />
  <title>{% block title %}My amazing blog{%
    endblock %}</title>
</head>
<body>
  {% block sidebar %}
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/blog/">Blog</a></li>
  </ul>
  {% endblock %}
  {% block content %}{% endblock %}
</body>
</html>
```

```
{% extends "base.html" %}

{% block title %}My amazing blog{%
  endblock %}

{% block content %}
  {% for entry in blog_entries %}
    <h2>{{ entry.title }}</h2>
    <p>{{ entry.body }}</p>
  {% endfor %}
{% endblock %}
```

load Tag

► *load Tag*: Loads a custom template tag set

► `{% load somelibrary package.otherlibrary %}`

► Ex.

► `<!DOCTYPE html>`

`{% load static %}`

`<html>`

`<head>`

`<link rel="stylesheet" type="text/css" href="{% static 'myapp/style.css' %}" />`

► More examples on:

<https://docs.djangoproject.com/en/4.2/howto/custom-template-tags/>

Static Files

- ▶ *Static files*: additional files e.g., images, JavaScript, or CSS needed to render the complete web page.
 - ▶ static files are placed in a folder under your app
 - ▶ e.g. `myapp/static/myapp/style.css`
 - ▶ Add `{% load static %}` at top of template file

```
<link rel="stylesheet" type="text/css" href="{% static 'myapp/style.css' %}"/>
```

`rel` = Specifies the relationship between the current document and the linked document

Shortcut function: `get_object_or_404`

- ▶ **`get_object_or_404(klass,*args,**kwargs)`**: Calls `get()` on a given model manager.
 - ▶ raises `Http404` instead of model's `DoesNotExist` exception.
 - ▶ Required arguments:
 - ▶ **`Klass`**: A Model class, a Manager, or a QuerySet instance from which to get the object.
 - ▶ **`**kwargs`**: Lookup parameters, which should be in the format accepted by `get()` and `filter()`.

An Example

```
from django.http import Http404
def my_view(request):
    try:
        my_object = MyModel.objects.get(pk=1)
    except MyModel.DoesNotExist:
        raise Http404
```

Alternatively,

```
from django.shortcuts import get_object_or_404
def my_view(request):
    my_object = get_object_or_404(MyModel, pk=1)
```

Other Shortcut Functions

- Visit the following URL:

<https://docs.djangoproject.com/en/4.2/topics/http/shortcuts/>

Class-Based Views?

References

- ▶ <https://docs.djangoproject.com/en/5.0/topics/templates/>
- ▶ <https://docs.djangoproject.com/en/5.0/topics/http/shortcuts/>
- ▶ <https://docs.djangoproject.com/en/5.0/howto/static-files/>
- ▶ Python Web Development with Django, by J. Forcier et al.
- ▶ Slides from Dr. Arunita and Dr. Saja